US006469799B1

(12) **United States Patent**
Kajita

(10) **Patent No.: US 6,469,799 B1**
(45) **Date of Patent: Oct. 22, 2002**

(54) **IMAGE FORMING APPARATUS AND IMAGE FORMING METHOD**

(75) Inventor: **Koji Kajita**, Yokohama (JP)

(73) Assignee: **Canon Kabushiki Kaisha**, Tokyo (JP)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/457,310**

(22) Filed: **Dec. 9, 1999**

(30) **Foreign Application Priority Data**

Dec. 11, 1998 (JP) 10-353566

(51) **Int. Cl.**[7] **G06K 1/00**
(52) **U.S. Cl.** **358/1.16**; 358/448; 382/309
(58) **Field of Search** 358/1.16, 448, 358/404, 452, 444, 409; 382/309, 295

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,526,128 A * 6/1996 Fujiki et al. 358/444

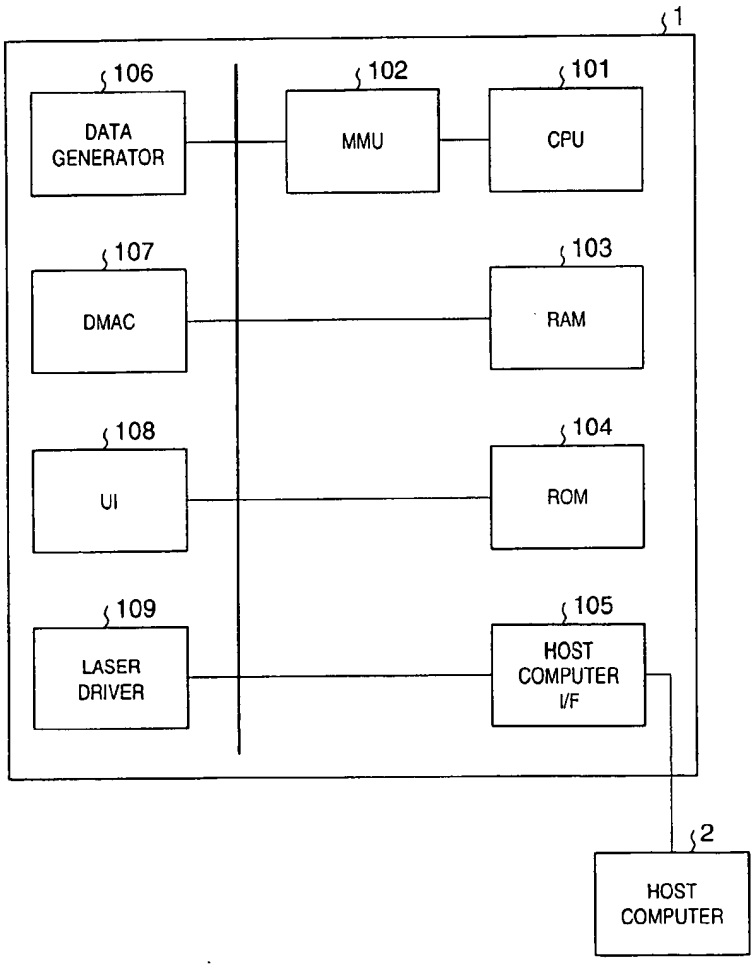
* cited by examiner

Primary Examiner—Mark Wallerson
Assistant Examiner—Twyler Lamb
(74) *Attorney, Agent, or Firm*—Fitzpatrick, Cella, Harper & Scinto

(57) **ABSTRACT**

Determination is made whether or not memory contents have been changed for each memory block. A physical memory corresponding to the memory block whose contents have not been changed is released as an area where no rendering data is stored. Then such area is managed distinguishably from other areas where rendering data are stored. As a result, memory areas are effectively utilized.

12 Claims, 6 Drawing Sheets



UNITED STATES PATENT AND TRADEMARK OFFICE

CERTIFICATE OF CORRECTION

PATENT NO. : 6,469,799 B1
DATED : October 22, 2000
INVENTOR(S) : Kajita

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Drawings:
Sheet 5 of 6, FIG. 5, "FLAME" should read -- FRAME --.

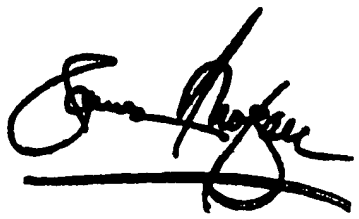
Column 1,
Line 26, "includes" should read -- include --.

Column 3,
Line 56, "more" should read -- More --; and
Line 65, "an" should read -- a --.

Column 6,
Line 55, "are" should read -- being --.

Signed and Sealed this

Twenty-second Day of April, 2003



JAMES E. ROGAN
Director of the United States Patent and Trademark Office

FIG. 1

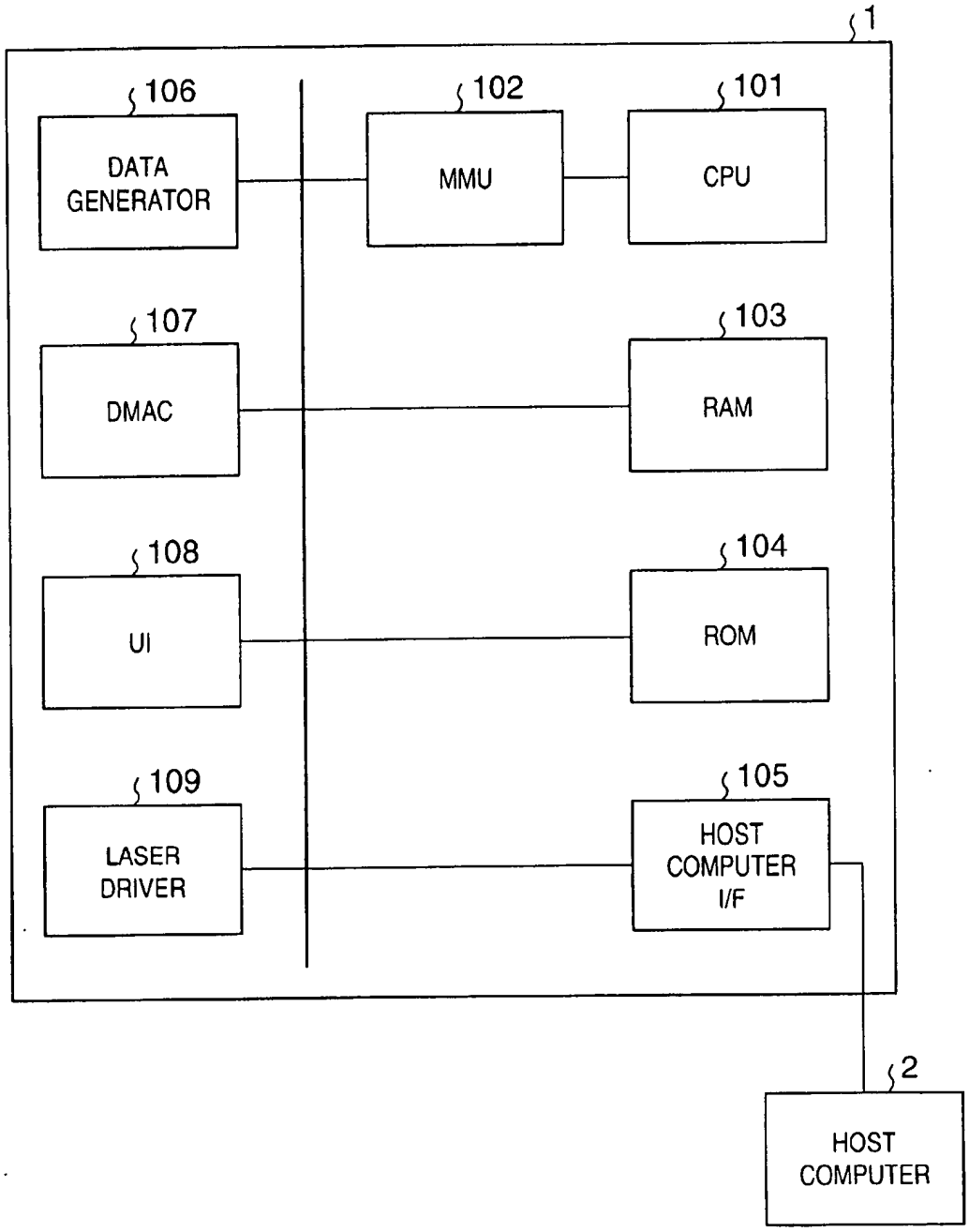

1
106
DATA GENERATOR
102
MMU
101
CPU
107
DMAC
103
RAM
108
UI
104
ROM
109
LASER DRIVER
105
HOST COMPUTER I/F
2
HOST COMPUTER

FIG. 2

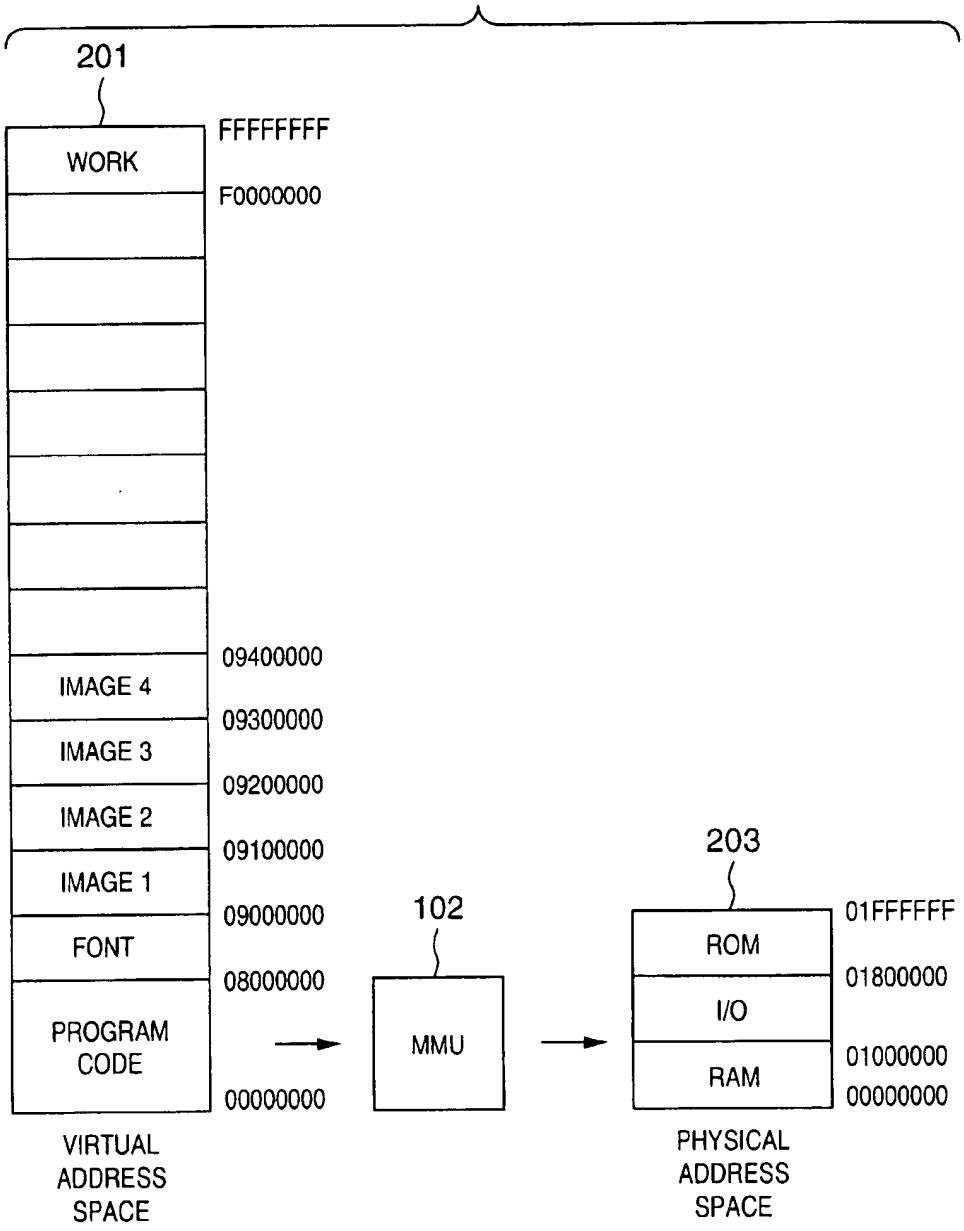

201
FFFFFFFF
WORK
F0000000
09400000
IMAGE 4
09300000
IMAGE 3
09200000
IMAGE 2
09100000
IMAGE 1
09000000
FONT
08000000
PROGRAM CODE
00000000
VIRTUAL ADDRESS SPACE
102
MMU
203
01FFFFFF
ROM
01800000
I/O
01000000
RAM
00000000
PHYSICAL ADDRESS SPACE

FIG. 3

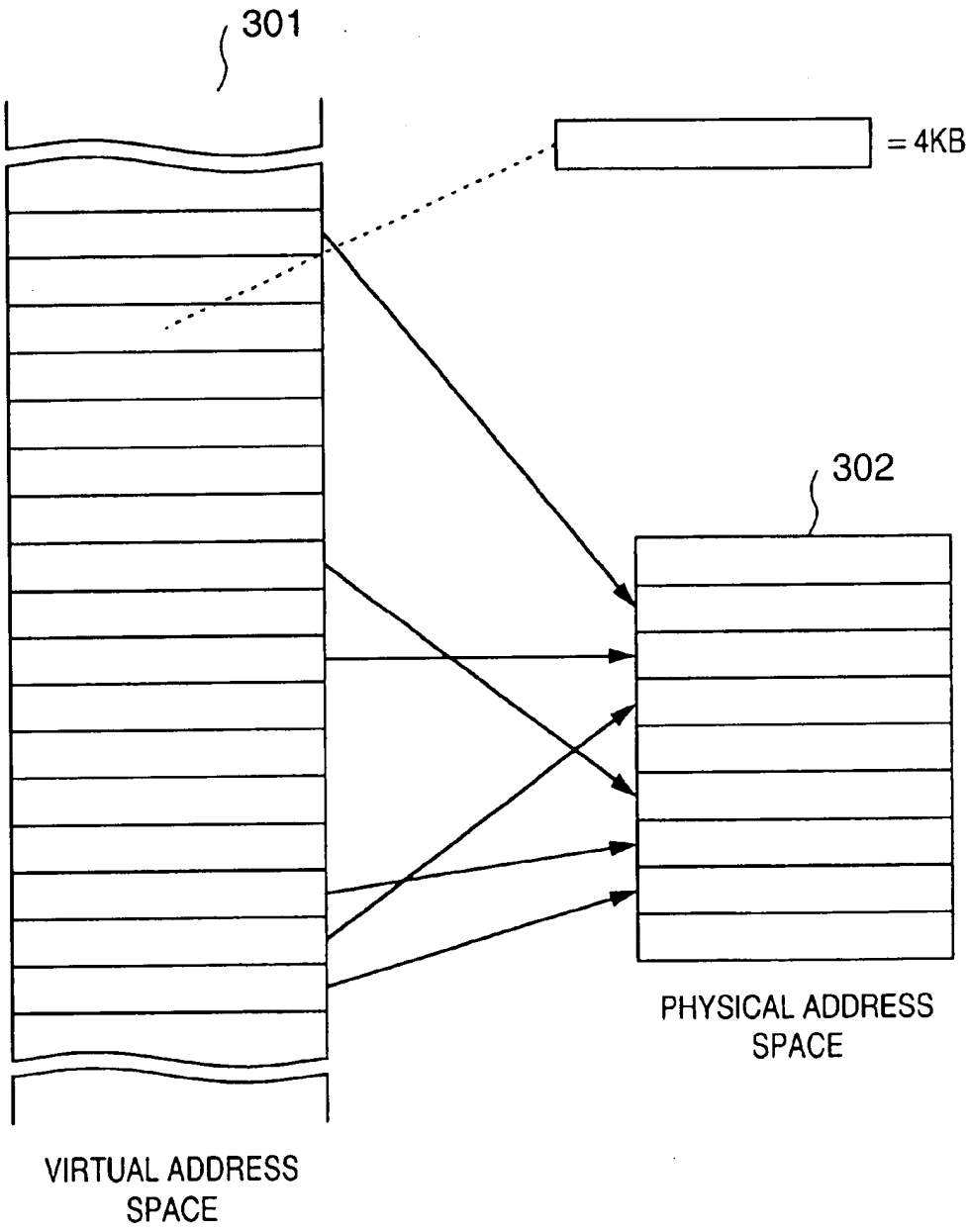
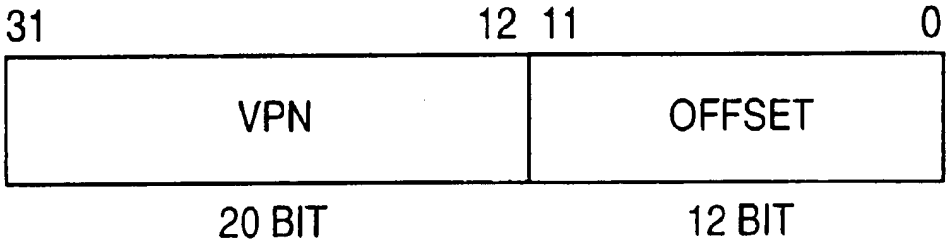


FIG. 4



VPN = VIRTUAL PAGE NUMBER

FIG. 5

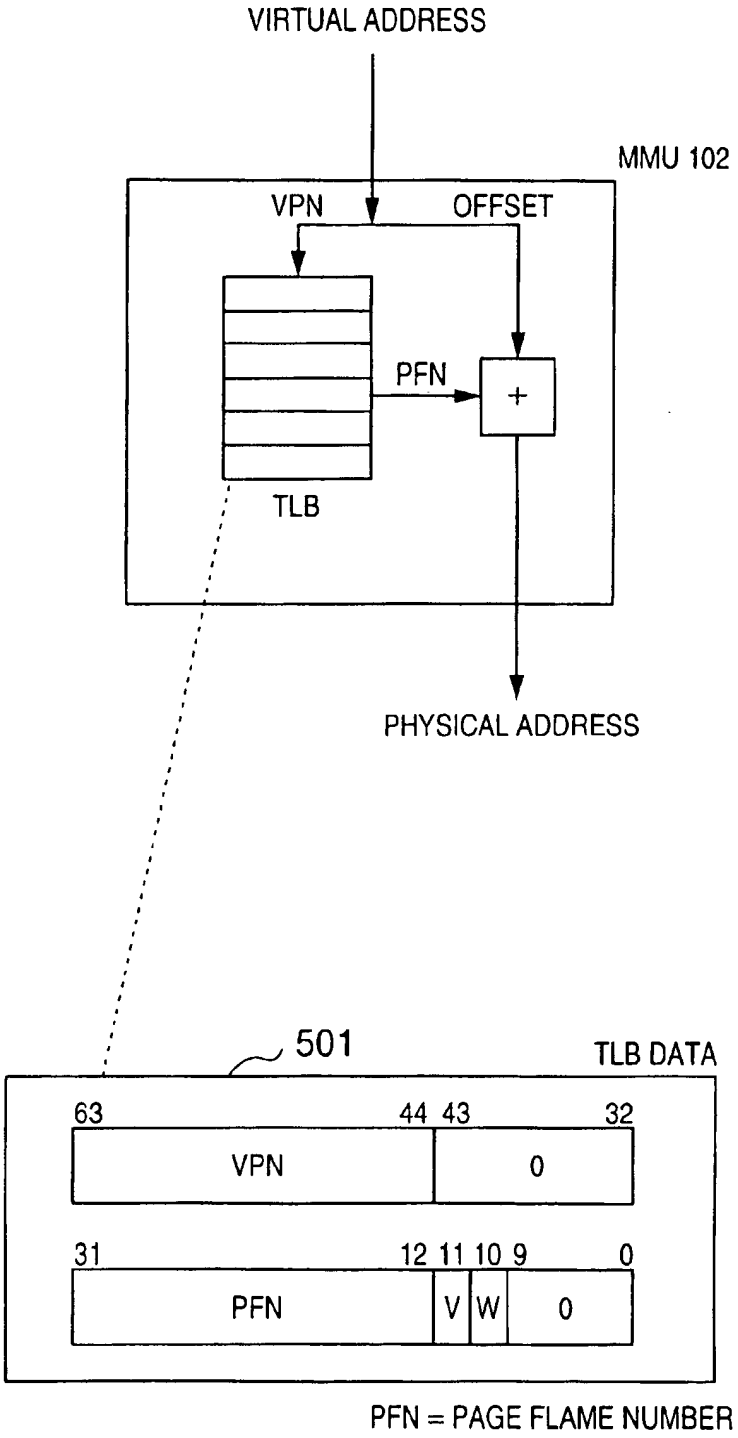

VIRTUAL ADDRESS
MMU 102
VPN
OFFSET
PFN
+
TLB
PHYSICAL ADDRESS
501
TLB DATA
63
44 43
32
VPN
0
31
12 11 10 9
0
PFN
V
W
0
PFN = PAGE FLAME NUMBER

FIG. 6

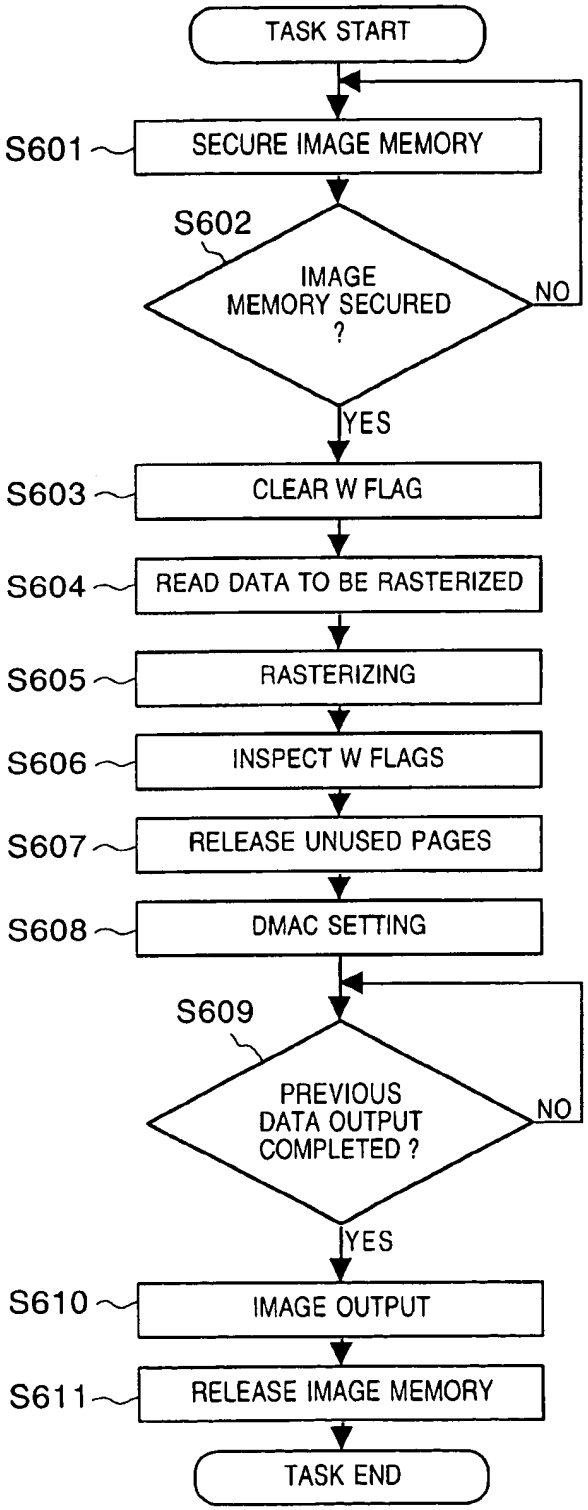

TASK START
S601
SECURE IMAGE MEMORY
S602
IMAGE MEMORY SECURED ?
NO
YES
S603
CLEAR W FLAG
S604
READ DATA TO BE RASTERIZED
S605
RASTERIZING
S606
INSPECT W FLAGS
S607
RELEASE UNUSED PAGES
S608
DMAC SETTING
S609
PREVIOUS DATA OUTPUT COMPLETED ?
NO
YES
S610
IMAGE OUTPUT
S611
RELEASE IMAGE MEMORY
TASK END

IMAGE FORMING APPARATUS AND IMAGE FORMING METHOD

BACKGROUND OF THE INVENTION

The present invention relates to an image forming apparatus, e.g., a laser-beam printer or the like, and image forming method for receiving data from a computer and performing printing on a medium, e.g., paper or the like, based on the received data.

Conventionally, a page printer, such as a laser-beam printer, employs an image formation method for supplying a printer with an image to be outputted which is expressed in a language independent of the printer resolution, and generating print data by a so-called rasterizing processing. Rasterizing processing is performed within the printer for converting the printer language into bitmap data corresponding to the resolution of the printer engine.

However, in the conventional rasterizing operation, it is necessary to store bitmap data, having a resolution corresponding to an output device, in a memory of the printer. Therefore, a memory area corresponding to the entire output page or a memory in a band unit corresponding to a part of the output page is necessary.

However, not all areas of an image to be outputted includes an object that needs rendering. Even in a case where output data mostly consists of blanks, the aforementioned memory area must be assigned. In view of this, efficient use of a memory area has been an issue that needs improvement.

SUMMARY OF THE INVENTION

The present invention is made in consideration of the above situation, and has as its object to provide an image forming apparatus and image forming method which can efficiently utilize a memory.

To achieve this object, the image forming apparatus and image forming method according to the present invention has the following configuration.

More specifically, the present invention provides an image forming apparatus for receiving inputted language describing an image to be outputted and generating a bitmap image for printing an image on a print medium, comprising: a physical memory for storing a program and data; a CPU for executing the program stored in the physical memory; a first address space for virtually storing data and a program executed when the CPU performs rasterizing operation; a second address space used for accessing the physical memory; address conversion means for performing mapping and management of address data from the first address space to the second address space in unit of a predetermined memory block; determination means for determining whether or not memory contents have been changed, in unit of the memory block managed by the address conversion means; and data supply means for supplying data, in place of the physical memory, when data is to be read out of the physical memory, wherein when an image is to be outputted, a physical memory corresponding to a memory block, whose contents have not been changed, is released according to a determination result of the determination means, and data generated by the data supply means is supplied when data is to be read out of the physical memory.

Furthermore, the present invention provides an image forming method of receiving inputted language describing an image to be outputted and generating a bitmap image for printing an image on a print medium, comprising: a step of storing a program and data in a physical memory; a step of causing a CPU to execute the program stored in the physical memory; a step of virtually storing in a first address space, data and a program executed when the CPU performs rasterizing operation; a step of accessing the physical memory by using a second address space; an address conversion step of performing mapping and management of address data from the first address space to the second address space in unit of a predetermined memory block; a determination step of determining whether or not memory contents have been changed, in unit of the memory block managed in the address conversion step; and a data supply step of supplying data, in place of the physical memory, when data is to be read out of the physical memory, wherein when an image is to be outputted, a physical memory corresponding to a memory block, whose contents have not been changed, is released according to a determination result of the determination step, and data generated at the data supply step is supplied when data is to be read out of the physical memory.

Moreover, according to an aspect of the present invention, the address conversion means extracts a corresponding page frame number from an associative memory based on a virtual page number stored in the first address space, and combines the extracted page frame number with offset data stored in the first address space to map address data to the second address space in unit of the memory block.

Moreover, according to an aspect of the present invention, the associative memory stores a virtual page number, page frame number, and flag information for identifying whether or not data contents have been changed.

Moreover, according to an aspect of the present invention, in the address conversion step, a corresponding page frame number is extracted from an associative memory based on a virtual page number stored in the first address space, and the extracted page frame number is combined with offset data stored in the first address space to map address data to the second address space in unit of the memory block.

Moreover, according to an aspect of the present invention, the associative memory stores a virtual page number, page frame number, and flag information for identifying whether or not data contents have been changed.

Moreover, according to an aspect of the present invention, the aforementioned image forming method further comprises the step of dividing an output image into a plurality of areas and rasterizing the plurality of areas when the CPU performs rasterizing operation.

Moreover, according to an aspect of the present invention, the aforementioned image forming apparatus further comprises: detection means for detecting same data or repetition of data when data is to be stored in the first address space; and storage means for storing a detection result of the same data or repetition of data in the address conversion means, wherein when an image is to be outputted, a physical memory corresponding to a memory block, having the same data or repetition of data, is released according to the detection result of the detection means, and data generated by the data supply means is supplied when data is to be read out of the physical memory.

Other features and advantages of the present invention will be apparent from the following description taken in conjunction with the accompanying drawings, in which like reference characters designate the same or similar parts throughout the figures thereof.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in and constitute a part of the specification, illustrate embodi-

ments of the invention, and together with the description, serve to explain the principles of the invention.

FIG. 1 is a block diagram showing an image generation circuit of an image forming apparatus according to the present invention;

FIG. 2 is a view for explaining the relationship between a virtual address space and physical address space in the image forming apparatus according to the present invention;

FIG. 3 shows memory allocation between a virtual address space and physical address space in the image forming apparatus according to the present invention;

FIG. 4 is an explanatory view of a virtual address structure according to the present invention;

FIG. 5 is an explanatory view showing a structure of a memory management unit (MMU) and a data structure in a TLB of the image forming apparatus according to the present invention; and

FIG. 6 is a flowchart showing steps of print-out operation by the image forming apparatus according to the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Preferred embodiments of the present invention will be described in detail in accordance with the accompanying drawings.

<First Embodiment>

FIG. 1 is a block diagram showing a printer construction according to the first embodiment of the present invention. In FIG. **1**, reference numeral **1** denotes a printer main body; and **2**, a host computer connected to the printer. The printer **1** is configured with a CPU **101**, a memory management unit (MMU) **102**, RAM **103**, ROM **104**, a host computer interface **105**, a data generator **106**, a DMA controller **107**, an operation panel interface **108**, and a laser driver **109**.

The CPU **101** controls respective units of the printer, and generates image data to be printed in accordance with instructions transmitted by the host computer **2**. The MMU **102** performs address conversion from a virtual memory address, used by the CPU for program execution, to a physical address where a physical memory exists. The RAM **103** is a random access memory which holds data such as programs or variables for operation of the CPU **101**, and holds image data of an image to be printed. The ROM **104** is a non-volatile memory which holds programs executed by the CPU **101** and other font data.

The host computer interface **105** communicates with the host computer **2**. The first embodiment employs a Centronics-compatible interface, capable of bi-directional communication and having a function to receive commands for printing or return printer status information to the host computer **2**. The data generator **106** generates print data, instead of the DMA controller reading data from the RAM **103**, when an image is to be printed. The DMA controller **107** transfers data. more specifically, the DMA controller **107** reads data out of the RAM **103** according to the setting of the CPU **101**, or reads data out of the data generator **106**, and transfers the read data to the laser driver **109**. The operation panel interface **108** performs various setting of the printer, and drives an operation panel for informing a user of a printer status. The laser driver **109** drives a laser diode (not shown) in accordance with data transmitted by the DMA controller **107**.

The printer **1** comprises an laser printer engine (not shown) and the laser printer engine prints an image by using the laser beam emitted from the laser diode.

FIG. 2 is a view for explaining the relationship of a memory space in the printer. Referring to FIG. **2**, reference numeral **201** denotes a virtual address space used when a program operates. The virtual address space includes areas for program codes or font data, work areas, and image data areas for storing bitmap data. The MMU **102** performs mapping of these virtual address areas to physical memory areas. By this, the CPU can access the physical address space **203** where physical memory exists. Hereinafter, processing in the image data areas particularly related to the present invention is described.

FIG. 3 is an enlarged schematic view of mapping in the image data areas. A virtual address space **301** includes consecutive memory areas, which are divided in pages of 4 KB block unit. In the actually allocated physical memory, non-consecutive page areas may be allocated as exemplified by reference numeral **302**.

FIG. 4 is an explanatory view of a virtual address structure. The CPU **101** has a 32-bit address. A virtual address consists of lower-order 12 bits allocated for an offset address (OFFSET) and higher-order 20 bits allocated for a virtual page number (VPN). The higher-order VPN is converted to a physical address by the MMU **102**. The lower-order OFFSET is used as a physical address, thus, 4 KB are allocated to a physical address, in unit of 12-bit blocks.

FIG. 5 is an explanatory view showing a structure of the MMU **102** and a data structure in a translation lookaside buffer (TLB). By utilizing an associative memory TLB provided for 64 entries, a physical address corresponding to a virtual address can be generated. The MMU **102** accesses TLB, storing the virtual page number (VPN) for the higher-order 20 bits of a virtual address, and acquires the page frame number (PFN) held therein.

The acquired PFN is combined with the OFFSET of the original virtual address, thereby converted to a physical address, and outputted.

Reference numeral **501** in FIG. **5** shows a data structure for one entry of the associative memory (TLB) included in the MMU **102**. The data structure for one entry having 64 bits consists of a virtual page number (VPN) of the virtual address, a page frame number (PFN) indicative of a physical address corresponding to the virtual address, and flag bits (V and W) each indicating a page status. When the CPU **101** accesses an address in the virtual address space, data is searched from data entries stored in the associative memory (TLB) having an address corresponding to the virtual page number (VPN) of the virtual address, and then the corresponding PFN value is obtained.

The MMU **102** combines the OFFSET with the PFN indicative of a physical address corresponding to the virtual page number (VPN), thereby obtaining an address in the physical memory.

The data structure in the TLB includes two flags V and W. V indicates that an entry is set in an effective page; and W is set when data is written in the effective page.

Note that when the CPU **101** refers to the TLB, if a page corresponding to the virtual address cannot be found in the data stored in the entries of the TLB, the MMU **102** informs the CPU of the occurrence of an exception. The CPU **101** then executes a memory management program in an exception processing, thereby updating the TLB entries.

In the exception processing, contents of entries which are not often used in the TLB entries are written back to a memory management table, and data corresponding to the virtual address is then written in the TLB entry.

FIG. 6 is a flowchart showing characteristic operation of a printer, as an example of an image forming apparatus,

according to the present invention. FIG. **6** explains a task implemented by the printer, which executes rasterizing of bitmap data for one screen and outputting an image. The task is generated for each output page. A plurality of rasterizing tasks are executed in parallel. Besides the rasterizing tasks, other tasks are also implemented in parallel, e.g., a task for receiving data by communicating with a host computer, a task as a pre-processing of rasterizing for translating data into a page description language so as to generate intermediate data and registering the data to a queue in page unit, and so forth.

Prior to rasterizing, an image memory area is allocated in step **S601** for storing rasterized bitmap data. More specifically, in step **S601**, a required memory capacity is requested to a virtual memory management program. The virtual memory management program then determines an available physical memory block and allocates consecutive memory areas in the virtual address space. The virtual memory management program also associates respective blocks with physical memory areas, updates the memory management table, and allocates the memory areas in the virtual address space as well as the physical memory areas. The allocated memory areas are simultaneously cleared to 0, thereby initialized to express a white blank.

In step **S602**, it is determined whether or not a memory area is allocated. If a memory area is not allocated (NO in **S602**), the control returns to step **S601** and loops until a memory area necessary for the task is released by other tasks and allocated.

If a memory area is allocated (YES in **S602**), the control proceeds to step **S603** where the W flag for the associative memory (TLB) data corresponding to the allocated physical memory and the W flag in the corresponding memory management table are initialized to 0.

In step **S604**, data to be rasterized is read from a data queue. The data stored in the data queue has been translated by a pre-processing program, in advance, from the data in a page description language transmitted by a host computer.

In step **S605**, bitmap data for an image is rendered in accordance with the read data, and outputted to the image memory area allocated in step **S601**. At this step, when data is written in a virtual address allocated for a physical address, the MMU **102** changes the W flag bit for this data to 1 in the TLB table. Therefore, once data is written in the physical address, 1 is set to the W flag for the TLB data of the corresponding block and to the W flag of a corresponding memory management table.

When rendering of bitmap data to be outputted is completed, all the W flags for the TLB data in the image memory areas and W flags in the memory management table are inspected in step **S606**. Then, blocks whose flag is still set to 0 ever since initialization in step **S603**, are listed.

In step **S607**, the physical memory allocation to the listed memory space is released so as to make the memory space available for other programs.

In step **S608**, the DMA controller **107** is set for outputting the rendered bitmap data. In order to output data consecutively arranged in the virtual address space, association between the virtual address space and the physical address space is made in advance by the CPU, and DMA operation (data generation) is performed by using a DMA table for data generation, in which the page order of the physical memory is prepared in advance.

The DMA table is set so that an address for the data generated by the data generator **106** is mapped as a physical address to an address allocated to the aforementioned released memory. By this, the DMA controller **107** reads 0 (white data), generated by the data generator **106**, instead of reading the released memory, and transfers it to the laser driver **109**. Accordingly, the same image data as that obtained in a case a memory is not released, can be transferred to the laser driver, while providing data generated by the generator **106**.

In step **S610**, the laser diode is driven, a latent image of an output image is formed on a photosensitive drum, and an image is printed on a print medium based on the latent image. When the image output is completed, the remaining pages in the allocated image memory are released in step **S611**, and data is removed from the memory management table, thereby making the memory areas available for other tasks. Then, the rasterizing task ends.

In the foregoing manner, rasterizing operation is performed and an image is printed.

<Second Embodiment>

In the first embodiment, the memory page, which has been once allocated but no data has been written, is released. A similar processing as that of the first embodiment can be applied in a case where each page contains uniform data. For instance, assuming a case of writing data in a memory, a mechanism for checking contents of data to be written may be provided to detect the same data or repeated data in all pages of a virtual address. If the same data or repeated data is detected, such information is added to the TLB data structure. By this, physical memory corresponding to the memory areas can be released prior to performing image output.

Furthermore, the second embodiment is described based on an assumption that a memory for forming an entire output image is allocated. However, in a case of dividing an output image into plural band areas and performing rasterizing, the present invention is applicable in the band unit.

<Other Embodiments>

The present invention can be applied to a system constituted by a plurality of devices (e.g., host computer, interface, reader, printer) or to an apparatus comprising a single device (e.g., copying machine, facsimile machine).

Further, the object of the present invention can also be achieved by providing a storage medium storing program codes for performing the aforesaid processes to a computer system or apparatus, reading the program codes, by a CPU or MPU of the computer system or apparatus, from the storage medium, then executing the program.

In this case, the program codes read from the storage medium realize the functions according to the embodiments, and the storage medium storing the program codes constitutes the invention.

Further, the storage medium, such as a floppy disk, a hard disk, an optical disk, a magneto-optical disk, CD-ROM, CD-R, a magnetic tape, a non-volatile type memory card, and ROM can be used for providing the program codes.

Furthermore, besides aforesaid functions according to the above embodiments are realized by executing the program codes which are read by a computer, the present invention includes a case where an OS (operating system) or the like working on the computer performs a part or the entire processes in accordance with designations of the program codes and realizes functions according to the above embodiments.

Furthermore, the present invention also includes a case where, after the program codes read from the storage medium are written in a function expansion card which is inserted into the computer or in a memory provided in a function expansion unit which is connected to the computer, CPU or the like contained in the function expansion card or

unit performs a part or the entire process in accordance with designations of the program codes and realizes functions of the above embodiments.

As has been set forth above, according to the present invention, contents of a physical memory corresponding to a virtual address space is substituted by another data generated by data generator. By virtue of this, the memory block can be released, making it possible to use the physical memory efficiently.

The present invention is not limited to the above embodiments and various changes and modifications can be made within the spirit and scope of the present invention. Therefore, to apprise the public of the scope of the present invention, the following claims are made.

What is claimed is:

1. An image forming apparatus for receiving inputted language describing an image to be outputted and generating a bitmap image for printing an image on a print medium, comprising:

a physical memory for storing a program and data;

a CPU for executing the program stored in the physical memory;

a first address space for virtually storing data and a program executed when said CPU performs rasterizing operation;

a second address space used for accessing said physical memory;

address conversion means for performing mapping and management of address data from said first address space to said second address space in unit of a predetermined memory block;

determination means for determining whether or not memory contents have been changed, in unit of the memory block managed by said address conversion means; and

data supply means for supplying data, in place of said physical memory, when data is to be read out of said physical memory,

wherein when an image is to be outputted, a physical memory corresponding to a memory block, whose contents have not been changed, is released according to a determination result of said determination means, and data generated by said data supply means is supplied when data is to be read out of said physical memory.

2. The image forming apparatus according to claim 1, wherein said address conversion means extracts a corresponding page frame number from an associative memory based on a virtual page number stored in said first address space, and combines the extracted page frame number with offset data stored in said first address space to map address data to said second address space in unit of the memory block.

3. The image forming apparatus according to claim 2, wherein the associative memory stores a virtual page number, page frame number, and flag information for identifying whether or not data contents have been changed.

4. The image forming apparatus according to claim 1, further comprising a laser printer engine.

5. The image forming apparatus according to claim 1, further comprising:

detection means for detecting same data or repetition of data when data is to be stored in said first address space; and

storage means for storing a detection result of the same data or repetition of data in said address conversion means,

wherein when an image is to be outputted, a physical memory corresponding to a memory block, having the same data or repetition of data, is released according to the detection result of said detection means, and data generated by said data supply means is supplied when data is to be read out of said physical memory.

6. An image forming method of receiving inputted language describing an image to be outputted and generating a bitmap image for printing an image on a print medium, comprising:

a step of storing a program and data in a physical memory;

a step of causing a CPU to execute the program stored in the physical memory;

a step of virtually storing in a first address space, data and a program executed when the CPU performs rasterizing operation;

a step of accessing the physical memory by using a second address space;

an address conversion step of performing mapping and management of address data from the first address space to the second address space in unit of a predetermined memory block;

a determination step of determining whether or not memory contents have been changed, in unit of the memory block managed in said address conversion step; and

a data supply step of supplying data, in place of the physical memory, when data is to be read out of the physical memory,

wherein when an image is to be outputted, a physical memory corresponding to a memory block, whose contents have not been changed, is released according to a determination result of said determination step, and data generated at said data supply step is supplied when data is to be read out of the physical memory.

7. The image forming method according to claim 6, wherein in said address conversion step, a corresponding page frame number is extracted from an associative memory based on a virtual page number stored in the first address space, and the extracted page frame number is combined with offset data stored in the first address space to map address data to said second address space in unit of the memory block.

8. The image forming method according to claim 7, wherein the associative memory stores a virtual page number, page frame number, and flag information for identifying whether or not data contents have been changed.

9. The image forming method according to claim 6, further comprising the step of dividing an output image into a plurality of areas and rasterizing the plurality of areas when the CPU performs rasterizing operation.

10. The image forming method according to claim 6, further comprising the steps of:

a step of detecting same data or repetition of data when data is to be stored in said first address space; and

a step of storing a detection result of the same data or repetition of data in said address conversion step,

wherein when an image is to be outputted, a physical memory corresponding to a memory block, having the same data or repetition of data, is released according to the detection result of said step of detecting, and data generated by said data supply step is supplied when data is to be read out of said physical memory.

11. An image processing apparatus comprising:

means for receiving first data specifying an image to be outputted;

means for converting the first data into second data, said converting means generates a first address for storing the converted second data into memory means;

address conversion means for converting said first address to second address;

determination means for determining whether or not memory contents have been changed, in unit of a memory block; and

data generating means for generating third data,

wherein said determination means determines whether or not the memory contents have been changed, when the memory contents have been changed, the second data are read out from said memory means and are supplied for outputting the image, when the memory contents have not been changed, the third data are generated by said data generating means and are supplied for outputting the image.

12. An image processing apparatus comprising:

means for receiving first data specifying an image to be outputted;

means for converting the first data into second data, said converting means generates a first address for storing the converted second data into memory means;

address conversion means for converting said first address to second address;

detection means for detecting repetition of same data in said second data;

storage means for storing a detection result of the repetition of the same data in said address conversion means; and

data generating means for generating the same data,

wherein when repetition of the same data is detected by said detection means, the same data which is generated by said data generating means is supplied for processing the image data, when repetition of the same data is not detected by said detection means, the second data is read out from said memory means, and is supplied for processing the image data.

* * * * *

US005684974A

United States Patent [19]

Onodera

[11] Patent Number: 5,684,974

[45] Date of Patent: Nov. 4, 1997

[54] METHOD AND APPARATUS FOR CONTROLLING RECONFIGURATION OF STORAGE-DEVICE MEMORY AREAS

[75] Inventor: Osamu Onodera, Handano, Japan

[73] Assignee: Hitachi, Ltd., Tokyo, Japan

[21] Appl. No.: 402,371

[22] Filed: Mar. 13, 1995

[30] Foreign Application Priority Data

Mar. 17, 1994 [JP] Japan 6-072536

[51] Int. Cl.[6] G06F 12/00; G06F 12/02; G06F 9/26; G06F 9/32

[52] U.S. Cl. 395/412; 395/415; 395/416; 395/419; 395/406; 395/427; 395/421.1

[58] Field of Search 340/172.5; 395/405, 395/412, 406, 416, 419, 415, 421.1, 417, 477.01

[56] References Cited

U.S. PATENT DOCUMENTS

3,723,976	3/1973	Alvarez et al.	340/172.5
4,075,694	2/1978	Ericsson	395/412
4,685,057	8/1987	Lemone et al.	395/421.1
4,761,737	8/1988	Duvall et al.	395/419

FOREIGN PATENT DOCUMENTS

03101275	2/1989	European Pat. Off.	G06F 9/46
2-33639	2/1990	Japan	12/2
2256513	12/1992	United Kingdom	G06F 12/02

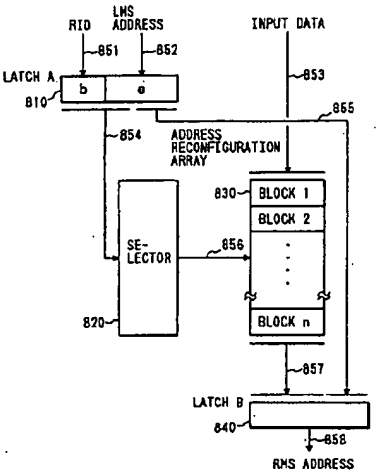
Primary Examiner—David K. Moore
Assistant Examiner—Than V. Nguyen
Attorney, Agent, or Firm—Antonelli, Terry, Stout & Kraus, LLP.

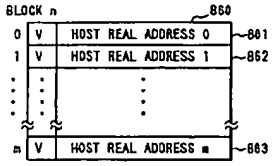
[57] ABSTRACT

An apparatus and method for controlling the reconfiguration of the physical storage area in a real storage device employed by an information processing system.

The invention includes an address reconfiguration array having a plurality of storage blocks which are each assigned to a virtual computer. Each storage block is composed of a plurality of host real-address entries. Assigned to a storage area in the logical memory of a virtual computer, each host real-address entry includes a validity field containing a validity bit and a host real-address field containing a high-order part of the start address of a real storage segment allocated to the storage area. The invention also includes a selector which receives the identifier of a virtual computer and a logical address from the virtual computer, and makes use of the identifier for choosing a storage block from the address reconfiguration array and a high-order portion of the logical address for selecting a host real-address entry from the chosen storage block. The value of the host real-address field of the selected host real-address entry, the high-order part of a real address, is then read out from the selected host real-address entry, and is merged with the low-order portion of the logical address in order to create a real address. If the validity bit indicates that the contents of the host real-address entry are invalid, however, the virtual computer is interrupted. A change to the contents of the address reconfiguration array can be made by replacing the contents of a host real-address entry chosen by the selector with update data.

8 Claims, 6 Drawing Sheets





Detail 8, 34

FIG. 1

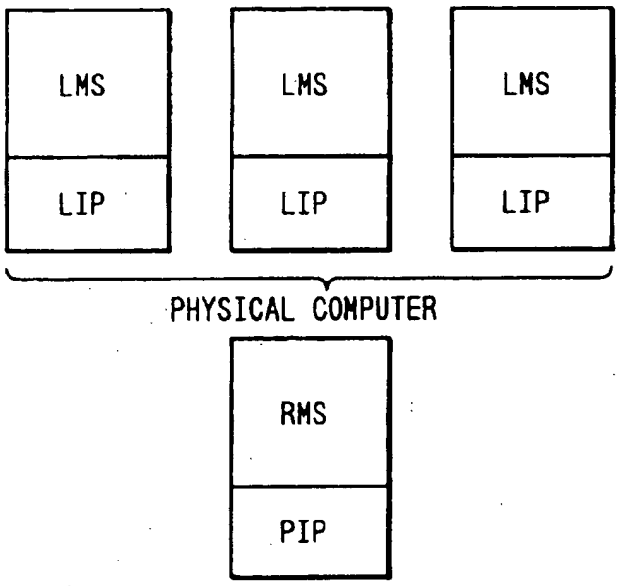

LMS
LMS
LMS
LIP
LIP
LIP
PHYSICAL COMPUTER
RMS
PIP

FIG. 2

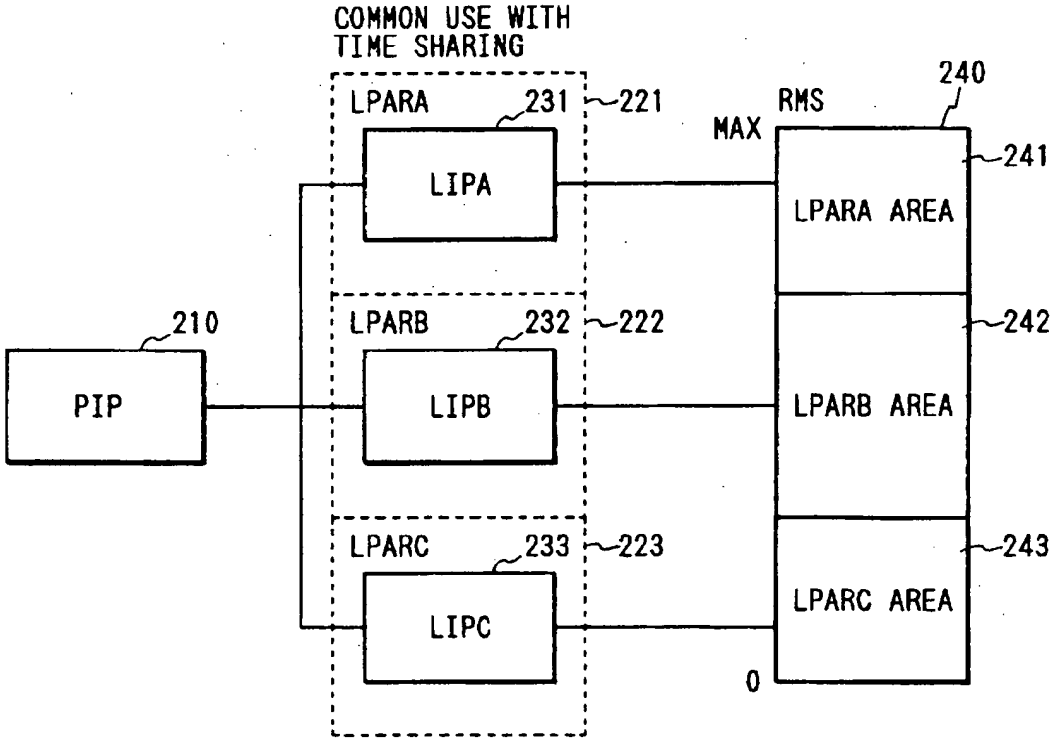

COMMON USE WITH TIME SHARING
LPARA
231
221
LIPA
LPARB
232
222
LIPB
LPARC
233
223
LIPC
210
PIP
240
RMS
MAX
241
LPARA AREA
242
LPARB AREA
243
LPARC AREA
0

FIG. 3(a)

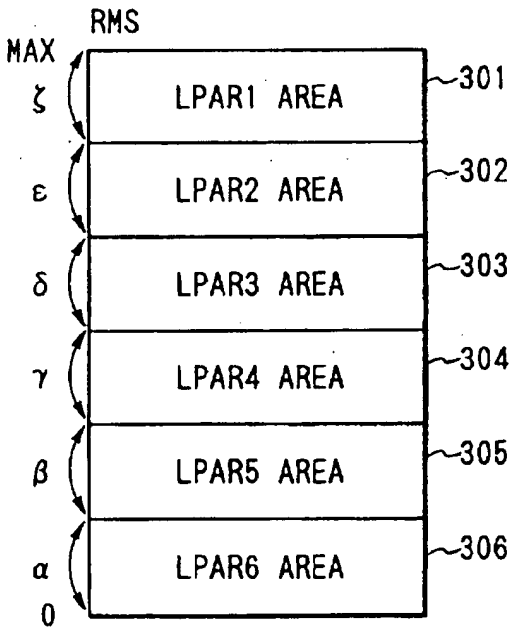

RMS
MAX
ζ
ε
δ
γ
β
α
0
LPAR1 AREA
LPAR2 AREA
LPAR3 AREA
LPAR4 AREA
LPAR5 AREA
LPAR6 AREA
301
302
303
304
305
306

FIG. 3(b)

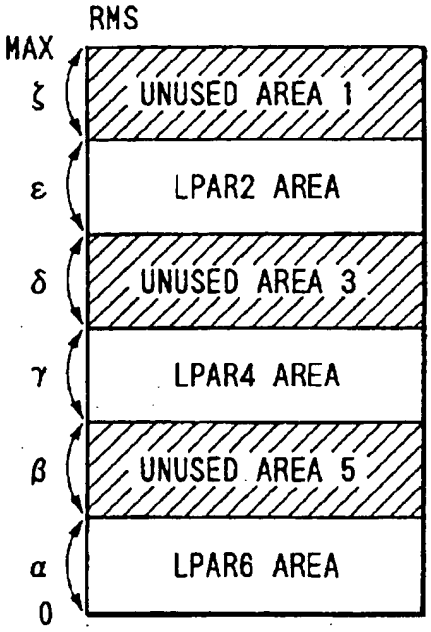

RMS
MAX
ζ
ε
δ
γ
β
α
0
UNUSED AREA 1
LPAR2 AREA
UNUSED AREA 3
LPAR4 AREA
UNUSED AREA 5
LPAR6 AREA

FIG. 4

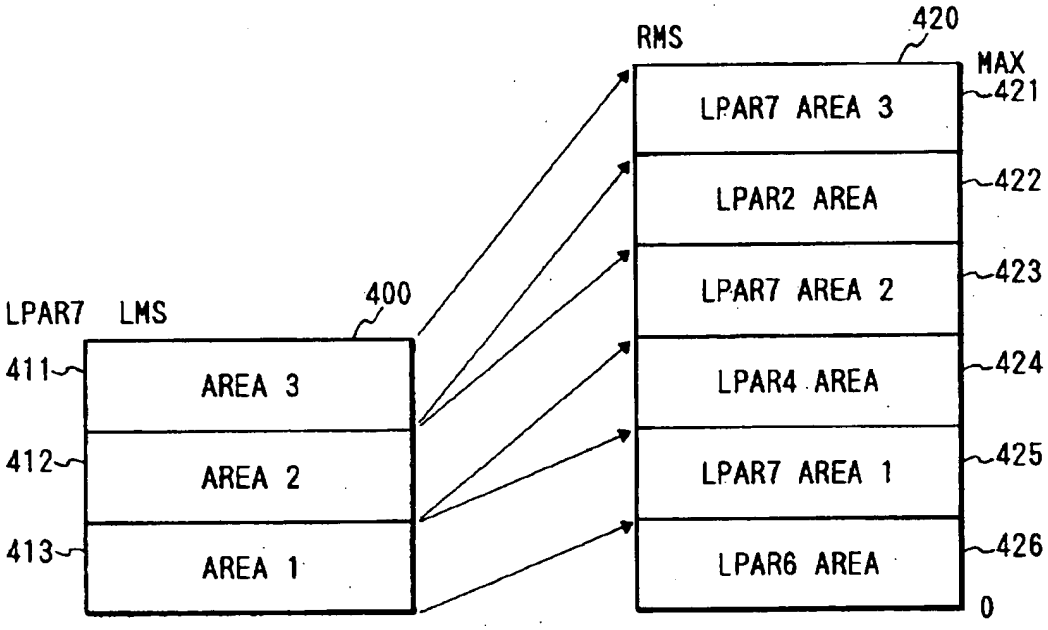

RMS
420
MAX
421
LPAR7 AREA 3
422
LPAR2 AREA
423
LPAR7 AREA 2
424
LPAR4 AREA
425
LPAR7 AREA 1
426
LPAR6 AREA
0
LPAR7 LMS
400
411
AREA 3
412
AREA 2
413
AREA 1

FIG. 5

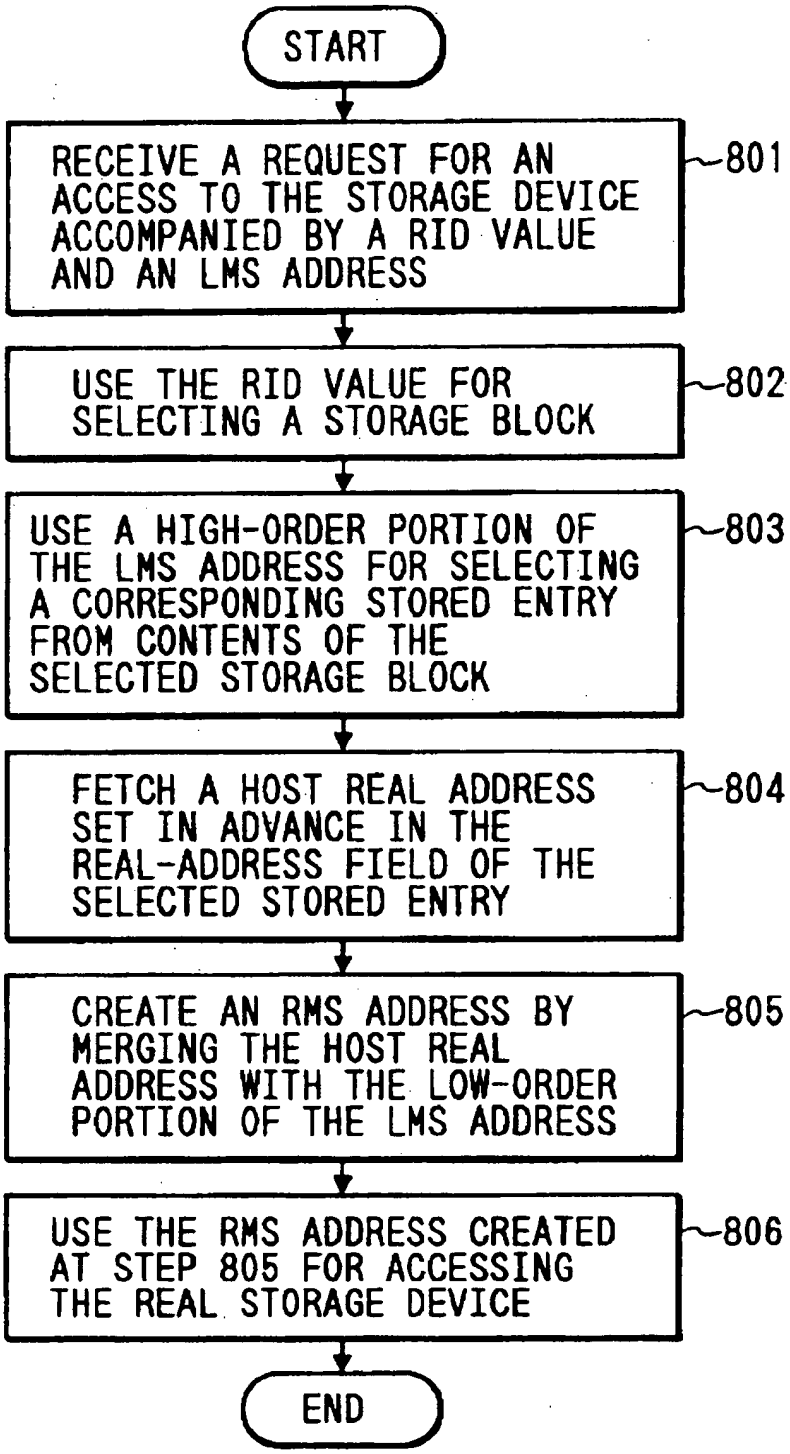

START
RECEIVE A REQUEST FOR AN ACCESS TO THE STORAGE DEVICE ACCOMPANIED BY A RID VALUE AND AN LMS ADDRESS
801
USE THE RID VALUE FOR SELECTING A STORAGE BLOCK
802
USE A HIGH-ORDER PORTION OF THE LMS ADDRESS FOR SELECTING A CORRESPONDING STORED ENTRY FROM CONTENTS OF THE SELECTED STORAGE BLOCK
803
FETCH A HOST REAL ADDRESS SET IN ADVANCE IN THE REAL-ADDRESS FIELD OF THE SELECTED STORED ENTRY
804
CREATE AN RMS ADDRESS BY MERGING THE HOST REAL ADDRESS WITH THE LOW-ORDER PORTION OF THE LMS ADDRESS
805
USE THE RMS ADDRESS CREATED AT STEP 805 FOR ACCESSING THE REAL STORAGE DEVICE
806
END

FIG. 6(a)

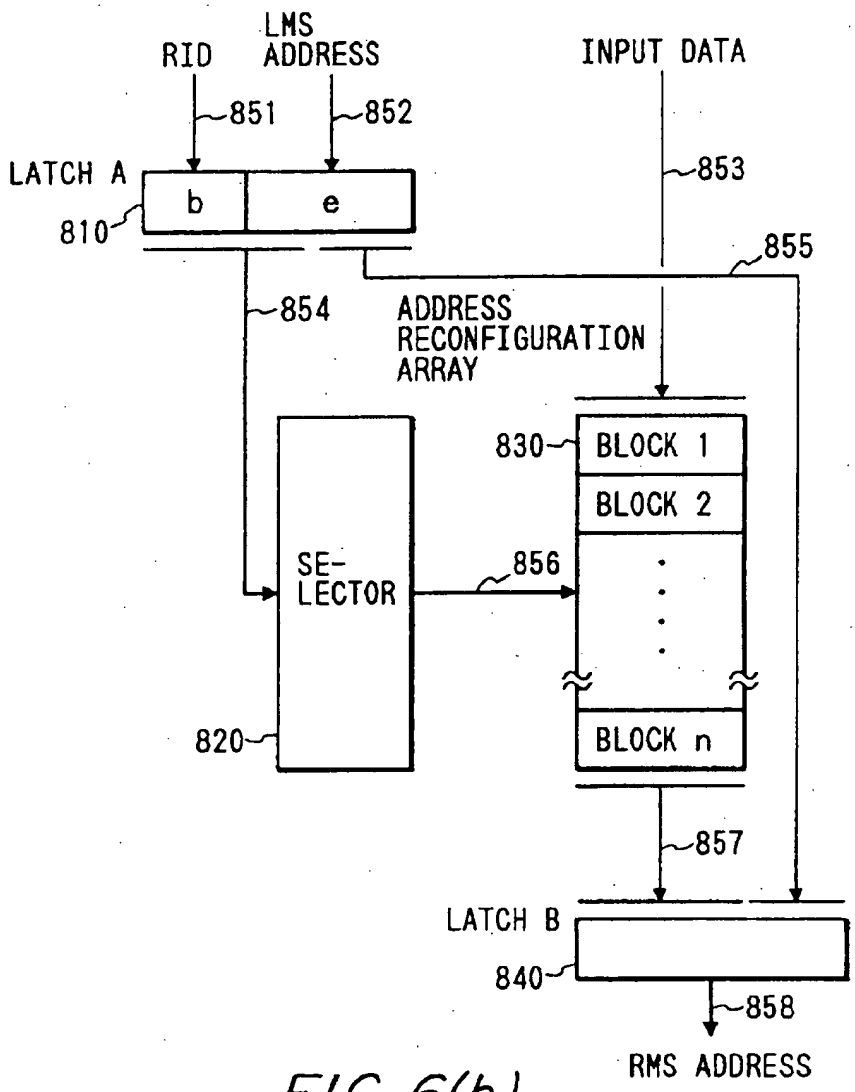

RID
LMS ADDRESS
INPUT DATA
851
852
853
LATCH A
b
e
810
855
854
ADDRESS RECONFIGURATION ARRAY
830
BLOCK 1
BLOCK 2
SE-LECTOR
856
820
BLOCK n
857
LATCH B
840
858
RMS ADDRESS

FIG. 6(b)

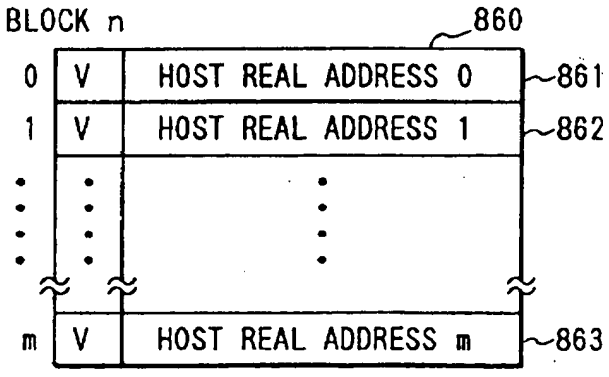
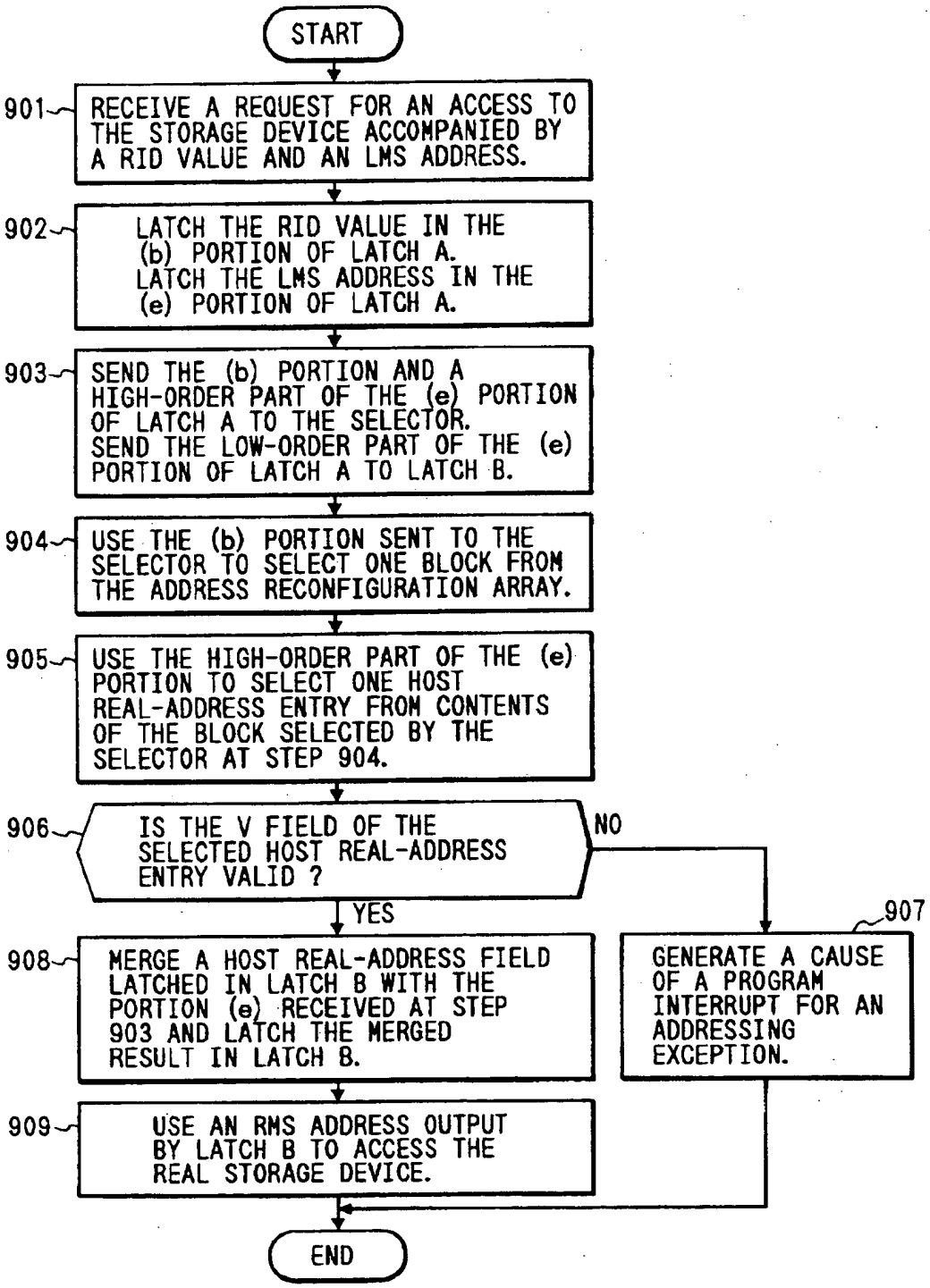

BLOCK n
860
0
V
HOST REAL ADDRESS 0
861
1
V
HOST REAL ADDRESS 1
862
m
V
HOST REAL ADDRESS m
863

FIG. 7


START
901
RECEIVE A REQUEST FOR AN ACCESS TO THE STORAGE DEVICE ACCOMPANIED BY A RID VALUE AND AN LMS ADDRESS.
902
LATCH THE RID VALUE IN THE (b) PORTION OF LATCH A. LATCH THE LMS ADDRESS IN THE (e) PORTION OF LATCH A.
903
SEND THE (b) PORTION AND A HIGH-ORDER PART OF THE (e) PORTION OF LATCH A TO THE SELECTOR. SEND THE LOW-ORDER PART OF THE (e) PORTION OF LATCH A TO LATCH B.
904
USE THE (b) PORTION SENT TO THE SELECTOR TO SELECT ONE BLOCK FROM THE ADDRESS RECONFIGURATION ARRAY.
905
USE THE HIGH-ORDER PART OF THE (e) PORTION TO SELECT ONE HOST REAL-ADDRESS ENTRY FROM CONTENTS OF THE BLOCK SELECTED BY THE SELECTOR AT STEP 904.
906
IS THE V FIELD OF THE SELECTED HOST REAL-ADDRESS ENTRY VALID ?
NO
YES
907
GENERATE A CAUSE OF A PROGRAM INTERRUPT FOR AN ADDRESSING EXCEPTION.
908
MERGE A HOST REAL-ADDRESS FIELD LATCHED IN LATCH B WITH THE PORTION (e) RECEIVED AT STEP 903 AND LATCH THE MERGED RESULT IN LATCH B.
909
USE AN RMS ADDRESS OUTPUT BY LATCH B TO ACCESS THE REAL STORAGE DEVICE.
END

METHOD AND APPARATUS FOR CONTROLLING RECONFIGURATION OF STORAGE-DEVICE MEMORY AREAS

BACKGROUND OF THE INVENTION

The present invention relates to conversion of logical storage-device memory addresses in virtual computers into real storage-device memory addresses in a physical computer employed in a virtual-computer system. In particular, the present invention relates to a technique for controlling the reconfiguration of storage-device memory areas in a process of allocating storage segments of a real storage device employed in a physical computer to logical storage areas in a plurality of virtual computers each having a logical storage device including such logical storage areas.

In general, a concept known as a virtual computer is embraced as a technique for running a plurality of operating systems on a single information processing apparatus. The virtual computer and operating system (OS) are also referred to hereafter as a logical partition (LPAR) and an OS respectively.

In order to implement LPARs on a single physical information processing apparatus, software called a VMCP (Virtual Machine Control Program) is executed on the information processing apparatus. A plurality of LPARs are created under the control of the VMCP. Furthermore, an independent OS is operated in each of the LPARs.

Thus, the VMCP has a function that allows a plurality of LPARs to share the single physical information processing apparatus which serves as a hardware resource in the virtual-computer system.

Methods for sharing the hardware resource provided by the single information processing apparatus among the LPARs include a technique of allocating the hardware resource on a time-sharing basis under the control of the VMCP, a technique of logically dividing the hardware resource into entities and allocating each entity exclusively to an LPAR, and a method of mixing the two above described techniques.

Next, the prior art is explained by referring to FIG. 1.

An example of a virtual-machine system is shown in FIG. 1. As shown in the figure, the virtual-machine system comprises three units of LPARs created in a single physical computer. The virtual-machine system is configured so that the three LPARs can operate independently of each other.

As shown in the figure, the physical computer comprises a real central processing unit and a real storage device which are also referred to hereafter as a PIP (Physical Instruction Processor) and an RMS (Real Main Storage) respectively. In the example, three LPAR systems are constructed to run on the single physical computer. The LPAR systems each comprise a logical central processing unit and a logical storage unit (logical memory) which are also referred to hereafter as an LIP (Logical Instruction Processor) and an LMS (Logical Main Storage) respectively. A VMCP runs on the PIP. The three LPAR systems operate independently of each other under the control of the VMCP. Processing functions of a LIP pertaining to an LPAR are implemented by allocating the hardware resource of the PIP on a time-sharing basis under the control of the VMCP. On the other hand, storage functions of a LMS pertaining to an LPAR are implemented by logically dividing the storage area of the RMS also under the control of the VMCP into storage segments and allocating each storage segment exclusively to an LPAR, or logically dividing a virtual storage area created on the RMS into portions and allocating each portion exclusively to an LPAR.

Relations among the PIP, LPARs, LIPs, LMSes and RMS are shown in FIG. 2.

The VMCP running on the PIP denoted by reference numeral **210** in the figure controls the allocation of the PIP **210**, a hardware resource, to virtual computers LPARA, LPARB and LPARC denoted by reference numerals **221**, **222** and **223** respectively. In addition to LMSes which are not shown in the figure, the LPARA **221**, LPARB **222** and LPARC **223** include logical central processing units LIPA, LIPB and LIPC respectively which are denoted by reference numerals **231**, **232** and **233** respectively.

The LIPA, LIPB and LIPC **231**, **232** and **233** operate independently of each other and also independently access the RMS which is denoted by reference numeral **240**.

The RMS **240** is logically divided into three storage segments which are associated with the LPARA, LPARB and LPARC **221**, **222** and **223** respectively. The three storage segments are LPARA, LPARB and LPARC areas denoted by reference numerals **241**, **242** and **243** respectively.

When the RMS **240** is divided into the three storage segments, the VMCP provides the LPARs with information on start addresses and storage sizes of the three storage segments allocated to the LPARs. In the virtual-machine system shown in FIG. **2**, the LIPA **231** can access only the LPARA area **241** while the LIPB **232** can access only the LPARB area **242**. Likewise, the LIPC **233** can access only the LPARC area **243**.

An LPAR actually accesses an LPAR area allocated thereto when any piece of software is executed on the LPAR after the LPAR is activated by the operator. It should be noted that activating a virtual computer is equivalent to powering up a physical computer.

The VMCP, which runs on the PIP, controlling the operations of the LPARs, can access all storage segments of the RMS **240**.

Next, an example of logically dividing the RMS into storage segments to be allocated to a plurality of LPARs is explained by referring to FIGS. **3**(*a*) and **3**(*b*).

An example of logically dividing the RMS into six storage segments and allocating each storage segment to an LPAR is shown in FIG. **3**(*a*). In this example, there are six LPARs: LPAR1, LPAR2, LPAR3, LPAR4, LPAR5 and LPAR6 to which an LPAR1 area **301**, an LPAR2 area **302**, an LPAR3 area **303**, an LPAR4 area **304**, an LPAR5 area **305** and an LPAR6 area **306** are allocated respectively. As described earlier, an LPAR area is actually accessed by the associated LPAR after the LPAR is activated.

The storage-area origin and storage size of the LPAR6 area **306** have values of 0 and α respectively. The storage-area origin and storage size are referred to hereafter as a STRORG (storage origin) and STREXT (storage extension) respectively. In other words, the LPAR6 area **306** is a storage segment in the RMS which starts at storage address 0 and ends at storage address $\alpha-1$.

The STRORG and STREXT values of the LPAR5 area **305** are α and β respectively. In other words, the LPAR5 area **305** is a storage segment in the RMS which starts α at storage address α and ends at storage address $\alpha+\beta-1$.

Similarly, the STRORG values of the LPAR4 area **304**, the LPAR3 area **303**, the LPAR2 area **302** and the LPAR1 area **301** are $\alpha+\beta$, $\alpha+\beta+\gamma$, $\alpha+\beta+\gamma+\delta$ and $\alpha+\beta+\gamma+\delta+\epsilon$ respectively. On the other hand, STREXT values of the LPAR4

area **304**, the LPAR3 area **303**, the LPAR2 area **302** and the LPAR1 area **301** are γ, δ, ϵ and ζ respectively.

FIG. **3**(*b*) shows a case in which some of the six LPARs assigned to storage segments of the RMS shown in FIG. **3**(*a*) have been activated.

Strictly speaking, all the six LPARs were once activated. Later on, the LPAR1, LPAR3 and LPAR5 were deactivated, leaving the LPAR2, LPAR4 and LPAR6 in an activated state as they are. FIG. **3**(*b*) is a diagram showing the state of allocation of storage segments in the RMS with the LPAR2, LPAR4 and LPAR6 remaining in an activated state.

As described above, FIG. **3**(*b*) shows the LPAR2, LPAR4 and LPAR6 in an activated state, using the LPAR2 area **302**, LPAR4 area **304** and LPAR6 area **306** of the RMS which have been allocated to them respectively.

Since the LPAR6 is activated, it is using the LPAR6 area **306** with STRORG and STREXT values of 0 and α respectively. As described above, the LPAR6 area **306** is a storage segment in the RMS which starts at address 0 and ends at address $\alpha-1$.

Since the LPAR5 is deactivated, on the other hand, the storage segment in the RMS starting at address α and ending at address $\alpha+\beta-1$ is not used.

Likewise, since the LPAR4 is activated, it is using the LPAR4 area **304** with STRORG and STREXT values of $\alpha+\beta$ and γ respectively. As described above, the LPAR4 area **304** is a storage segment in the RMS which starts at address $\alpha+\beta$ and ends at address $\alpha+\beta+\gamma-1$. Similarly, since the LPAR2 is activated, it is using the LPAR2 area **302** with STRORG and STREXT values of $\alpha+\beta+\gamma+\delta$ and ϵ respectively. As described above, the LPAR2 area **302** is a storage segment in the RMS which starts at address $\alpha+\beta+\gamma+\delta$ and ends at address $\alpha+\beta+\gamma+\delta+\epsilon-1$.

Much like the LPAR5, since the LPAR3 is deactivated, the storage segment in the RMS starting at address $\alpha+\beta+\gamma$ and ending at address $\alpha+\beta+\gamma+\delta-1$ is not used. Likewise, since the LPAR1 is deactivated, the storage segment in the RMS starting at address $\alpha+\beta+\gamma+\delta+\epsilon$ and ending at address $\alpha+\beta+\gamma+\delta+\epsilon+\zeta-1$ is not used. Accordingly, the sizes of the unused storage segments in the RMS in this case are β, δ and ζ.

Here, an attempt is made, for example, to activate an LPAR7 requiring a storage capacity of η. In this case, η is compared to the sizes of the unused storage segments: β, δ and ζ. The LPAR7 can be activated only if η is found smaller than or equal to the largest among β, δ and ζ. If η is greater than β, δ and ζ, on the other hand, the LPAR7 cannot be activated.

The LPAR7 cannot be activated even if the total amount of two or three of β, δ and γ is equal to or greater than η. This is because β, δ and γ are the sizes of noncontiguous storage segments. Since a contiguous unused storage segment having a size equal to or greater than η does not exist in the RMS, the LPAR7 cannot be activated.

As described above, with the conventional technique of the prior art to logically divide the RMS into storage segments and allocate these storage segments to LPARs, a new LPAR requiring a storage capacity greater than the size of the largest contiguous unused storage segment available in the RMS cannot be activated even if the total size of all unused storage segments existing in the RMS is greater than the storage capacity. Accordingly, the unused storage segments in the RMS with a total size exceeding the storage capacity required the LPAR to be activated cannot be used, because they are not contiguous. The conventional technique has a problem that it may be impossible to activate a new LPAR only because a contiguous unused storage segment having a size equal to or greater than the storage capacity required by the new LPAR does not exist in the RMS even if the total size of all mused storage segments available in the RMS is greater than the required storage capacity. From the system-operational point of view,. this problem cannot be ignored.

A conventional technology adopted to solve the problem described above is disclosed in Japanese Patent Laid-open No. 2-33639 with a title 'Main-Memory Management of a Virtual Machine System.'

A memory-management system according to the conventional technology described in 2-33639 includes:

- start-address registers for storing start addresses of all storage segments in the real main storage device of a physical computer allocated to logical storage areas of the logical memory of each activated LPAR;
- boundary-address registers for storing boundary addresses of logical storage areas of the logical memory employed in each activated LPAR;
- a comparator for determining one of the logical storage areas in a logical memory which a logical address in the logical memory pertains to; and
- an adder for converting the logical address into a real address in the real main storage device by adding the logical address to a value determined by the contents of a start-address register assigned to a storage segment in the real main storage device allocated to the logical storage area determined by the comparator.

Accordingly, a plurality of noncontiguous storage segments of the real main storage device can be allocated to storage areas of the logical memory of an LPAR as if the storage segments were a contiguous resident area, making it possible to reserve storage segments with any arbitrary sizes, which satisfy the capacity of the logical memory employed in an LPAR, in the real main storage device at a high speed. As a result, a logical address in the logical memory of an LPAR can thus be converted into a real address in the real main storage device of the physical computer at a high speed with a real start address stored in a start-address register used as a base. On top of that, the conventional technique allows a virtual-machine system to be implemented, wherein the real main storage device can be allocated to a plurality of LPARs effectively without interrupting operations of already activated LPARs.

With the technique of the conventional technology described in 2-33639 to logically divide the RMS into storage segments and allocate the storage segments to a plurality of LPARs, however, at least as many start-address registers and as many boundary-address registers as storage segments having noncontiguous addresses in the RMS are required. As the number of such storage segments increases, a large number of registers, adders and comparators are also required as well.

If signal lines connecting these registers, adders and comparators to each other are implemented by hardware logic, much hardware logic also needs to be added, entailing an extremely high cost of the information processing apparatus which gives rise to an industrial problem in the process of manufacturing of industrial products. This industrial problem is so serious that it cannot be ignored by all means.

SUMMARY OF THE INVENTION

It is an object of the present invention to provide a real storage device that keeps the manufacturing cost low, enhances the operatability of the system and, at the same

time, allows the real storage device to be utilized efficiently through the implementation of area reconfiguration of the real storage device for preventing the numbers of required registers, adders and comparators from rising substantially due to a larger number of noncontiguous segments becoming available among a plurality of storage segments in the real storage device.

In order to achieve the object described above, the present invention provides a method for controlling reconfiguration of the physical storage area of a real storage device employed in an information processing apparatus employing a central processing unit in addition to the real storage device. The physical storage area is logically divided into a plurality of storage segments to be allocated to a plurality of logical memories which are each employed in one of a plurality of virtual computers all operating on the central processing unit.

Utilized in the method is an address reconfiguration array which includes a plurality of storage blocks each assigned to one of the virtual computers. As described above, the virtual computers each have a logical memory. The logical memory is divided into a plurality of logical storage areas by a high-order portion of logical addresses in the logical memory. Each of the storage blocks is composed of a plurality of host real-address entries which are each assigned to one of the logical storage areas and used for storing a high-order part of the start address of a storage segment. Also used in this method in addition to the address reconfiguration array are selecting apparatus for selecting one of the storage blocks from the address reconfiguration array in accordance with a virtual computer's identifier issued by the virtual computer and selecting one of the host real-address entries from the selected storage block in accordance with the high-order portion of a logical address specified also by the same virtual computer. When a virtual computer makes a request for conversion of a logical address in the logical memory of the virtual computer into a real address in the real storage device by issuing the identifier of the logical computer and the logical address to the selecting apparatus, the identifier is used by the selecting apparatus for choosing one of the storage blocks from the address reconfiguration array whereas the high-order portion of the logical address is used by the selecting means for selecting one of the host real-address entries from the selected storage block. The high-order part of the start address of a storage segment read out from the selected host real-address entry is then merged with the remaining low-order portion of the logical address to create the real address in the storage segment of the real storage device. When a virtual computer makes a request to change the contents of a host real-address entry in the address reconfiguration array, the virtual computer issues its identifier and a logical address in its logical memory to the selecting apparatus as well as the high-order part of the start address of a storage segment to the address reconfiguration array. The identifier is used by the selecting apparatus for choosing one of the storage blocks from the address reconfiguration array while the high-order portion of the logical address is used by the selecting apparatus for selecting one of the host real-address entries from the selected storage block. The high-order part of the start address of the storage segment is then written into the selected host real-address entry, replacing teh previous contents of the selected host real-address entry.

In addition, each of the host real-address entries comprises a validity field for storing a flag indicating whether or not the contents of the host real-address entry are valid, and a host real-address field for storing the high-order part of the start address a storage segment. When the contents of a selected host real-address entry are read out in processing a request made by one of the virtual computers for address conversion, the flag in the validity field is examined to determine whether or not the contents of the selected host real-address entry are valid. If the contents of selected host real-address entry are found invalid, a cause of a program interrupt indicating an addressing exception is reported to the virtual computer making the request. If the contents of the selected host real-address entry are found valid, on the other hand, the requested address conversion is carried out and an address resulting from the address conversion is used as a valid real address in the real storage device. When a request to make a change to the contents of one of the host real-address entries selected by the request from the address reconfiguration array is made by one of the virtual computers, validity information and the high-order part of the start address of a storage segment issued to the address reconfiguration array along with the request are used to replace present values of the validity and host real-address fields of the selected hose real-address entry respectively.

It should be noted that, if only validity information is issued to the address reconfiguration array in a request made by one of the virtual computers to make a change to the contents of one of the host real-address entries selected by the request from the address reconfiguration array, only the validity field of the selected host real-address entry is updated. If only the high-order part of the start address of a storage segment is issued to the address reconfiguration array in a request made by one of the virtual computers to make a change to the contents of one of the host real-address entries selected by the request from the address reconfiguration array, on the other hand, only the host real-address field of the selected host real-address entry is updated.

According to the present invention, a host real-address entry is selected from the address reconfiguration array after a virtual computer is activated. The high-order part of the start address of a storage segment is then read out from the selected host real-address entry and merged with the remaining low-order portion of a logical address specified by the activated virtual computer, allowing a real address in the storage segment of the real storage device to be created in an address-generation process. In an update process, on the other hand, an address specified by the activated virtual computer can then be written into the selected host real-address entry, replacing previous contents of the host real-address entry: the high-order part of the start address of a storage segment.

As described above, a host real-address entry comprises two fields, i.e. a validity field and a host real-address field. In a process of generating a real address in a storage segment of the real storage device, the real address is generated only if a flag stored in the validity field denotes that the contents of a selected host real-address entry are valid. If the flag stored in the validity field indicates that the contents of the selected host real-address entry are invalid, on the other hand, a cause of a program interrupt indicating an addressing exception is reported to a virtual computer requesting the generation of the real address. In a process of making a change to the contents of the address reconfiguration array, a host real-address entry is selected from the address reconfiguration array, and pieces of information stored in both the validity and host real-address fields of the selected host real-address entry or either one of the fields are updated in accordance with an instruction issued by a virtual computer making a modification request.

In this way, control of the reconfiguration of storage areas, which remarkably enhances the utilization efficiency of the real storage device, can be implemented at a low cost.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a conceptual diagram showing a configuration of sharing a physical processor among a plurality of virtual computers in accordance with the conventional technology;

FIG. 2 is a conceptual diagram showing a configuration of a plurality of virtual computers operating on a single physical processor and a plurality of storage segments of a real storage device allocated to the virtual computers in accordance with the conventional technology;

FIG. 3(*a*) is a diagram showing allocation of storage segments of a real storage device to a plurality of virtual computers;

FIG. 3(*b*) is a diagram showing allocation of storage segments of a real storage device to a plurality of virtual computers, wherein some of the virtual computers are not using storage segments allocated to them, in accordance with the conventional technology;

FIG. 4 is a diagram showing typical allocation of noncontiguous unused storage segments of a real storage device to contiguous logical storage segments of a virtual computer wherein more than one noncontiguous unused storage-area segments are allocated to the contiguous logical memory of the virtual computer.

FIG. 5 is a flow chart briefly showing a procedure for converting a memory access to a logical storage area of a contiguous LMS into a memory access to a real storage segment of a noncontiguous RMS adopted by an embodiment of the present invention;

FIG. 6(*a*) is a logic block diagram showing a hardware configuration of an embodiment provided by the present invention;

FIG. 6(*b*) is a diagram showing a detailed configuration of one of storage blocks composing an address reconfiguration array; and

FIG. 7 is a flow chart showing in detail the procedure for converting a memory access to a logical storage area of a contiguous LMS into a memory access to one of noncontiguous storage segments of the RMS adopted by the embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention will become apparent from the following detailed description of preferred embodiments of a technique for allocating memory areas of a storage device adopted in a system for controlling the reconfiguration of the memory areas with reference to accompanying diagrams.

FIG. 4 is a diagram showing an activated virtual machine LPAR7 specifying a required storage capacity η to which the RMS' three noncontiguous unused storage segments **5**, **3** and **1** shown in FIG. 3(*b*) are to be typically allocated. The three noncontiguous unused storage segments are shown in FIG. 4 as LPAR7 areas **1** to **3** respectively.

As shown in FIG. 3(*b*), the unused storage segment **5** starts at storage address α and ends at storage address $\alpha+\beta-1$ whereas the unused storage segment **3** starts at storage address $\alpha+\beta+\gamma$ and ends at storage address $\alpha+\beta+\gamma+\delta-1$. As for the unused storage segment, the start and end storage addresses are $\alpha+\beta+\gamma+\delta+\epsilon$ and $\alpha+\beta+\gamma+\delta+\epsilon+\zeta-1$ respectively. Accordingly, the sizes of the unused storage segments **5**, **3** and **1** are β, δ and ζ respectively.

Let the storage capacity η required by the LPAR7 be equal to the sum of β, δ and ζ in this example. In this case, an LMS **400** pertaining to the LPAR7 requiring the contiguous storage capacity η is divided into three logical storage areas LMS1, LMS2 and LMS3 denoted by reference numerals **413**, **412** and **411** respectively in FIG. 4 so that the LMS1, LMS2 and LMS3 logical storage areas **413**, **412** and **411** have a capacity β, δ and ζ respectively. The storage capacities β, δ and ζ are deliberately made the same as the sizes of the LPAR7 areas **1** to **3** of the RMS denoted by reference numerals **425**, **423** and **421**, allowing the LPAR7 areas **425**, **423** and **421** to be allocated to the LMS1, LMS2 and LMS3 to form well-fitting memory mapping. In other words, the LMS **400** of the LPAR7 requiring the contiguous storage capacity η is divided into the logical areas **413**, **412** and **411** which are exactly mapped to the noncontiguous storage segments **425**, **423** and **421** respectively.

At the time the LPAR7 specifying the required storage capacity η is activated, only part of the storage capacity η is used. In some cases, part of the storage capacity η is put in an off-line state some time after the activation. In this case, the storage capacity ζ or the storage capacities δ and ζ, for example, are treated as if their values were zero. It should be noted that a storage capacity with a value of zero implies that its associated storage segment in the RMS is unallocated.

FIG. 5 is a flowchart of a procedure for converting a memory access to a logical storage area in the contiguous LMS into a memory access to one of noncontiguous storage segments in the RMS for a case in which the contiguous LMS is divided into logical storage areas mapped to noncontiguous storage segments of the RMS.

First of all, at a step **801** of the flowchart, an LIP makes a request to access its contiguous LMS, issuing an RID (Region ID) indicating the identification of the LPAR employing the LIP and a logical address at which the LMS is to be accessed. When this request is made, the contiguous LMS of the LPAR has already been divided into logical storage areas mapped to three noncontiguous storage segments in the RMS. The start addresses of the three storage segments in the RMS and logical addresses in the LMS mapped to the start addresses have been stored in hardware logic circuits in advance.

At a step **802**, the RID is used to select a storage block indicated by the RID. The storage block comprises a plurality of host real-address entries each having a host real-address field for storing a start address in the RMS to which the logical address received at the step **801** is mapped.

At a step **803**, a high-order portion of the logical address received from the LIP is used to select a host real-address entry from the storage block selected at the step **802**.

At a step **804**, a high-order portion of an RMS address corresponding to the a high-order portion of the logical address specified at the step **801** is fetched from the host real-address field of the host real-address entry selected at the step **803**.

At a step **805**, the high-order portion of the RMS address corresponding to the high-order portion of the logical address obtained at the step **804** is merged with the remaining low-order portion of the logical address to produce a real address.

At a step **806**, the RMS is accessed at the real address produced at the step **805**.

The above description has briefly explained the procedure for converting a memory access to a logical storage area in the contiguous LMS into a memory access to one of noncontiguous storage segments in the RMS.

Next, a detailed embodiment provided by the present invention for implementing the above conversion process is described by referring to FIGS. 6(*a*) and (*b*) as well as FIG. 7.

FIG. 7 is a flowchart showing in detail the procedure for converting a memory access to a logical storage area of a contiguous LMS into a memory access to one of noncontiguous storage segments of an RMS adopted by the embodiment. FIG. 6(*a*) is a logic block diagram showing a hardware configuration of the embodiment for implementing the conversion procedure shown in FIG. 7 whereas FIG. 6(*b*) is a diagram showing a detailed configuration of one of storage blocks composing an address reconfiguration array.

As shown in FIG. 6(*a*), an RID representing contiguous logical storage areas in the LMS is input through a signal line **851** which is connected to a latch A denoted by reference numeral **810**. In addition, a logical storage address in the contiguous LMS is input through a signal line **852** which is also connected to the latch A **810** as well.

Having its inputs connected to the signal lines **851** and **852**, the latch A **810** serves as a relay register for temporarily storing the RID representing the contiguous logical storage areas in the LMS received through the signal line **851** and the logical storage address in the contiguous LMS received through the signal line **852**. The latch A **810** is connected to a selector **820** and a latch B **840** by signal lines **854** and **855** respectively.

The selector **820** receives the RID representing the contiguous logical storage areas in the LMS and a high-order portion of the logical storage address in the contiguous LMS output by the latch A **810** through the signal line **854**. The selector **820** is used for selecting one among a plurality of storage blocks (block **1** to block n) composing an address reconfiguration array **830**.

The selector **820** is also connected to the address reconfiguration array **830** by a signal line **856**. As described above, the address reconfiguration array **830** includes a plurality of storage blocks. One of the storage blocks is selected by a block select command signal transmitted through the signal line **856**. The contents of a host real-address entry in the selected storage block are transmitted to the latch B **840** through a signal line **857**.

Data is written into a host real-address entry in a storage block selected from the address reconfiguration array **830** through a signal line **853** also connected to the address reconfiguration array **830**.

Having its inputs connected to the signal lines **857** and **855**, the latch B **840** serves as a relay register for temporarily storing the contents of a host real-address entry in a selected one among the storage blocks composing the address reconfiguration array **830** and the remaining low-order portion of a logical address in the contiguous LMS transmitted through the signal lines **857** and **855** respectively. The latch B **840** outputs an RMS address in the real storage device through a signal line **858**.

The above description explains a detail hardware-logic configuration of an embodiment implementing a procedure for converting a memory access to a logical storage area of a contiguous LMS into a memory access to one of noncontiguous real storage segments of the RMS.

Next, details of a logic configuration of the address reconfiguration array **830** are explained by referring to FIGS. 6(*a*) and (*b*).

As shown in FIG. 6(*a*), the address reconfiguration array **830** comprises a plurality of storage blocks. One of the storage blocks composing the address reconfiguration array **830** is selected in accordance with a block select command signal transmitted through the signal line **856**.

As shown in FIG. 6(*b*), any selected storage block **860** further comprises a plurality of host real-address entries, entry **0**, entry **1** to entry m denoted by reference numerals **861**, **862** to **863** respectively. One of the host real-address entries **861** to **863** (entry **0** to entry m) composing a selected storage block is selected in accordance with an entry select command signal also received through the signal line **856**. Each of the host real-address entries **861** to **863** comprises two fields. One of the field is a V (validity) field containing a flag for indicating whether or not the contents of host real-address entry are valid. The other one is a host real-address field for storing the start address of a storage segment in the RMS.

If the flag in the V field indicates that the host real-address entry is valid, the contents of its host real-address field are output to the B latch **840** through the signal line **857**. If the flag in the V field indicates that the contents of host real-address entry are invalid, on the other hand, a signal is output to indicate that a cause of a program interrupt for an addressing exception exists.

In this example, a host real-address entry of a storage block **860** of the address reconfiguration array **830** represents 1M (mega) bytes of memory and a storage block **860** accommodates 2,064 host real-address entries. In other words, a storage block **860** can be used to handle addresses of up to 2 G (giga) bytes of memory. In this case, the two G (giga) bytes are the capacity of the RMS.

As described above, data can be written into a host-real address entry of a selected storage block **860** in the address reconfiguration array **830** through the signal line **853** connected thereto. Such data can be stored in any arbitrary host real-address entry in one among a plurality of storage blocks **860** composing the address reconfiguration array **830**.

The above description explains a detailed embodiment of an address reconfiguration array serving as a principal part of hardware used to implement a procedure for converting a memory access to a logical storage area of a contiguous LMS into a memory access to one of noncontiguous storage segments of the RMS.

Next, details of the address-conversion process are described by referring to FIGS. 6(*a*) and (*b*) as well as FIG. 7.

As shown in FIG. 7, the procedure for converting a memory access to a logical storage area of a continuous LMS into a memory access to one of noncontiguous storage segments of the RMS comprises steps **901** to **909**. A conversion processing procedure for each step is explained as follows.

First of all, at the step **901**, an LIP makes a request to access its contiguous LMS, issuing a predetermined RID (Region ID) assigned to an LPAR in operation which the LIP pertains to. In addition to the predetermined RID, the LIP also specifies a logical address, at which the LMS is to be accessed. The predetermined RID and the logical address are temporarily stored in the A latch **810** through the signal lines **851** and **852** respectively.

When this request is made, the contiguous LMS of the LPAR has already been divided into logical storage areas mapped to noncontiguous storage segments in the RMS. Host real-address entries of a storage block in the address reconfiguration array **830** which is assigned to the LPAR have each already been filled with information on the mapping of a logical storage area in the contiguous LMS of the LPAR to one of noncontiguous storage segments in the RMS.

At the step **902**, the predetermined RID received through the signal line **851** is latched in the (b) portion of the A latch **810** whereas the logical address received through the signal line **852** is latched in the (e) portion of the A latch **810**.

Having its inputs connected to the signal lines **851** and **852**, the A latch **810** serves as a relay register for temporarily storing the predetermined RID received through the signal line **851** and the logical address in the contiguous LMS received through the signal line **852**.

At the step **903**, the predetermined RID latched in the (b) portion of the A latch **810** and a high-order part of the logical address latched in the (e) portion of the A latch **810** are transmitted to the selector **820** through the signal line **854**. At the same time, the remaining low-order part of the logical address latched in the (e) portion of the A latch **810** is supplied to the B latch **840** through the signal line **855**.

At the step **904**, the selector **820** forwards the predetermined RID, which has been received through the signal line **854** at the step **903**, to the address reconfiguration array **830** through the signal line **856**, requesting the address reconfiguration array **830** to select one of a plurality of storage blocks composing the address reconfiguration array **830**. Receiving the predetermined RID through the signal line **856**, the address reconfiguration array **830** selects a storage block accordingly.

At the step **905**, the selector **820** forwards the high-order part of the logical address, which has also been received through the signal line **854** at the step **903**, to the address reconfiguration array **830** through the signal line **856**, requesting the address reconfiguration array **830** to select one of a plurality of host real-address entries composing the storage block selected at the step **904**. Receiving the high-order part of the logical address through the signal line **856**, the address reconfiguration array **830** selects a host real-address entry accordingly. The address reconfiguration array **830** then transmits the contents of the selected host real-address entry to the B latch **840** through the signal line **857**.

At the step **906**, the B latch **840** receives through the signal line **857** the contents of the host real-address entry selected at the step **905** from a plurality of host real-address entries composing the storage block selected at the step **904** from a plurality of storage blocks constituting the address reconfiguration array **830** and latches them therein. The B latch **840** then examines the flag in the V field of the host real-address entry to see whether or not the contents of the host real-address entry are valid. If the host real-address entry is found valid, the flow of processing continues to the step **908**. If the host real-address entry is found invalid, on the other hand, the flow of processing continues to the step **907**.

At the step **907**, a cause of a program interrupt signaling an addressing exception is generated. In this case, the operation to access the real storage device is suspended. As described above, the processing at this step is executed only if the flag in the V field of the host real-address entry indicates that the contents of the host real-address entry are invalid. The 'invalid' value of the flag in the V field of the host real-address entry implies that either no storage segment of the RMS is allocated to the logical storage area of the LMS being accessed or the storage segment of RMS is in an off-line state.

At the step **908**, the contents of the host real-address entry selected at the step **905** from a plurality of host real-address entries composing the storage block selected at the step **904** from a plurality of storage blocks composing the address reconfiguration array **830**, which have been transmitted through the signal line **857**, are latched in a high-order portion of the B latch **840**. In addition, the remaining low-order part of the logical address in a storage area of the contiguous LMS, which has been transmitted through the signal line **855**, is latched in the remaining low-order portion of the B latch **840**. As described above, the processing at this step is executed only if the flag in the V field of the host real-address entry indicates that the contents of the host real-address entry are valid.

That is to say, at this step, the contents of the host real-address field in a host real-address entry are treated as a high-order part of an RMS address whereas the remaining low-order part of the logical address in a storage area of the contiguous LMS, which has been transmitted through the signal line **855**, is used as the remaining low-order part of the RMS address. By merging both the high and low-order parts with each other, a complete RMS address can be created. This RMS address is an address in the real storage device at which the RMS is to be actually accessed.

At the step **909**, a request is made to access the RMS (real storage device) at the real storage (RMS) address created at the step **908**.

The above description explains details of an embodiment for implementing the procedure for converting a memory access to a logical storage area of a contiguous LMS into a memory access to one of noncontiguous storage segments of the RMS.

Next, an embodiment implementing a procedure for changing at run time the contents of a host real-address entry in one of a plurality of storage blocks composing the address reconfiguration array **830** is described by referring FIGS. **6**(*a*) and (*b*).

When an LIP makes a request to make a change at run time to the contents of a host real-address entry in one of a plurality of storage blocks composing the address reconfiguration array **830**, its RID and a logical address in a storage area of the contiguous LMS are specified in the request. The RID and the logical address are received through the signal lines **851** and **852** respectively. The RID received through the line **851** is latched in the (b) portion of the A latch **810** whereas the logical address received through the signal line **852** is latched in the (e) portion of the A latch **810**.

Latching the RID and the logical address received through the signal lines **851** and **852** respectively, the A latch **810** transmits the RID latched in the (b) portion thereof and a high-order part of the logical address latched in its (e) portion to the selector **820** through the signal line **854**.

The selector **820** in turn forwards the RID in the (b) portion, which has been received through the signal line **854**, to the address reconfiguration array **830** through the signal line **856**, requesting the address reconfiguration array **830** to select one of a plurality of storage blocks composing the address reconfiguration array **830**.

Receiving the RID through the signal line **856**, the address reconfiguration array **830** selects a storage block accordingly.

As for the (e) portion, the selector **820** forwards the high-order part of the logical address, which has also been received through the signal line **854**, to the address reconfiguration array **830** through the signal line **856**, requesting the address reconfiguration array **830** to select one of a plurality of host real-address entries composing the selected storage block.

Receiving the high-order part of the logical address through the signal line **856**, the address reconfiguration array **830** selects a host real-address entry accordingly. Update-data received by the address reconfiguration array **830** through the signal line **853** is then written into the selected host real-address entry.

By carrying out at run time the operation to write update-data into a series of host real-address entries such as the one described above, RMS addresses corresponding to logical addresses in storage areas of any contiguous LMS owning an RID can be changed arbitrarily at run time. On top of that, by changing the flag in the V field of a host real-address entry from a 'valid' value to an 'invalid' one, the logical storage area assigned to the host real-address entry can be disconnected at run time from the LMS. By changing the flag in the V field of a host real-address entry from an 'invalid' value to a 'valid' one, on the contrary, the logical storage area assigned to the host real-address entry can be connected at run time to the LMS.

As described above, according to the present invention, if a contiguous unused storage segment, which has a size equal to or larger than the storage capacity required by an LPAR, does not exist in the RMS for allocation to the LPAR upon the activation thereof, start addresses of noncontiguous unused storage segments existing in the RMS are set in a plurality of host real-address entries composing a storage block in the address reconfiguration array **830** allocated to the LPAR, allowing the logical storage areas in the LMS of the LPAR to be mapped to the noncontiguous unused storage segments. The contents of the host real-address entries composing a storage block in the address reconfiguration array **830** are used in an address-conversion process, allowing a logical address in a storage area of the contiguous LMS specified by the LPAR into a real address in one of the noncontiguous storage segments of the RMS. That is to say, the present invention provides a means for allocating any unused storage segments of the real storage area of the RMS, even if the unused storage elements are noncontiguous, to logical storage areas of an LMS as though the unused storage segments were contiguous.

After the RMS is allocated to an LPAR, an OS operating on the LPAR may disconnect or connect storage areas of its LMS by using an off-line or on-line command. In order to do it, the V fields of a plurality of host real-address entries stored in the hardware are changed at run time from a 'valid' value to an 'invalid' one or vice versa. In this way, the mapping and the sizes of a plurality of allocated storage segments in the RMS can be changed at run time. That is to say, when an OS operating on the LPAR disconnects a storage area of its LMS by using an off-line command once a storage segment of the RMS has been allocated to that storage area, the storage segment of the RMS allocated to the disconnected storage area of the LMS can be reconfigured at run time from a used state to an available (or unused) one.

When the V field of a host real-address entry stored in memory in the hardware is changed at run time from a 'valid' value to an 'invalid' one or vice versa, a real address stored in the host real-address field of the same host real-address entry can also be updated as well. The mapping of an LMS address to an RMS address can thus be changed at run time too.

As a result, the operatability of the system is enhanced. On top of that, a method of controlling the reconfiguration of storage segments of a real storage device, which increases the utilization efficiency of the storage device, can be implemented.

In this embodiment, a method of controlling the reconfiguration of storage segments of a real storage device implemented by hardware logic is adopted. It should be noted, however, that the configuration of such storage segments can also be controlled by a micro program as well.

While keeping the cost low, what have been discovered in the present invention can be used to implement the conversion of an address in a logical memory of a virtual computer into an address in a real storage device, set as well as modify the mapping of a plurality of storage areas in a logical memory to a plurality of storage segments in a real storage device and validate as well as invalidate the mapping. On top of that, the present invention allows the operatability of the system to be enhanced and, at the same time, the utilization efficiency of the real storage device to be increased as well.

I claim:

1. A method of controlling reconfiguration of a physical storage area of a real storage device included in an information processing apparatus having said real storage device, and a plurality of virtual computers operating on a central processing unit, wherein said physical storage area is logically divided into a plurality of storage segments to be allocated to a plurality of logical memories each employed in one of said virtual computers operating on said central processing unit, and an address reconfiguration array which includes a plurality of storage blocks each being assigned to one of said virtual computers and having its logical memory divided into a plurality of logical storage areas according to a high-order portion of logical addresses thereof, wherein each of said storage blocks is composed of a plurality of host real-address entries each being assigned to one of said logical storage areas and used for storing a high-order part of a start address of one of said storage segments of said physical storage area, said method comprising the steps of:

selecting a storage block from said address reconfiguration array in accordance with an identifier of a virtual computer, said identifier being issued by said virtual computer;

selecting one of said host real-address entries from the selected storage block in accordance with said high-order portion of a logical address specified by said virtual computer which issued said identifier; and

generating a real address of said real storage device by reading out said high-order part of a start address of one of said storage segments from the selected host real-address entry, and merging said high-order part of said start address of one of said storage segments with a remaining low-order portion of said logical address to create said real address of said storage segment of said real storage device.

2. A method according to claim **1**, wherein each host real-address entry comprises a validity field for storing a flag indicating whether or not contents of said host real-address entry are valid, and a host real-address field for storing said high-order part of a start address of one of said storage segments of said real storage device; and

wherein said generating step comprises the steps of:

examining said flag in said validity field to determine whether or not said contents of said selected host real-address entry are valid,

reporting, when said contents of the selected host real-address entry are found invalid, a cause of a program interrupt indicating an addressing exception to said virtual computer making said request, and

carrying out, when said contents of the selected host real-address entry are found valid, said request for address conversion and using an address resulting from address conversion as a valid real address of said real storage device.

3. A method according to claim **2** wherein said generating step further comprises the steps of:

updating, when only validity information is issued to said address reconfiguration array in said request for address conversion to make a change to contents of one of said host real-address entries selected by said request for address conversion, only said validity field of the selected host real-address entry; and

updating, when only said high-order part of a start address of one of said storage segments is issued to said address reconfiguration array in said request for address conversion to make a change to contents of one of said host real-address entries selected by said request for address conversion, only said host real-address field of the selected host real-address entry.

4. An apparatus for controlling reconfiguration of a physical storage area of a real storage device included in an information processing system having said real storage device and a plurality of virtual computers operating on a central processing unit, wherein said physical storage area is logically divided into a plurality of storage segments to be allocated to a plurality of logical memories each employed in one of said virtual computers operating on said central processing unit, said apparatus comprising:

an address reconfiguration array which includes a plurality of storage blocks each being assigned to one of said virtual computers and having its logical memory divided into a plurality of logical storage areas according to a high-order portion of logical addresses thereof, wherein each of said storage blocks is composed of a plurality of host real-address entries each being assigned to one of said logical storage areas and used for storing a high-order part of a start address of one of said storage segments of said physical storage area;

a selector for selecting a storage block from said address reconfiguration array in accordance with an identifier of a virtual computer, said identifier being issued by said virtual computer and selecting one of said host real-address entries from the selected storage block in accordance with a high-order portion of a logical address specified by said virtual computer which issued said identifier; and

an address generator for generating a real address of said real storage device by reading out said high-order part of a start address of one of said storage segments from the selected host real-address entry and merging said high-order part of a start address of one of said storage segments with a remaining low-order portion of said logical address to create said real address of said storage segment of said real storage device.

5. An apparatus according to claim 4, further comprising:

modification means for making a change to contents of the selected host real-address entry by replacing previous contents of the selected host real-address entry.

6. An apparatus according to claim 4, wherein each of said host real-address entries comprises:

a validity field for storing a flag indicating whether or not contents of said host real-address entry are valid; and

a host real-address field for storing said high-order part of a start address of one of said storage segments.

7. A method according to claim 1, further comprising the step of:

making a change to contents of the selected host real-address entry by replacing previous contents of the selected host real-address entry.

8. A method according to claim 2, further comprising the step of:

making a change to contents of the selected host real-address entry by replacing previous contents of the validity and host real-address fields of the selected host real-address entry.

* * * * *

US 20020105523A1

(19) United States

(12) Patent Application Publication (10) Pub. No.: US 2002/0105523 A1

Behrbaum et al. (43) Pub. Date: Aug. 8, 2002

(54) METHOD AND SYSTEM FOR ALLOCATING MEMORY FROM THE LOCAL MEMORY CONTROLLER IN A HIGHLY PARALLEL SYSTEM ARCHITECTURE (HPSA)

(76) Inventors: **Todd S. Behrbaum**, Issaquah, WA (US); **Ronald T. Horan**, Houston, TX (US); **Stephen R. Johnson JR.**, Redmond, WA (US); **John E. Theisen**, Conroe, TX (US)

Correspondence Address:
CONLEY ROSE & TAYON, P.C.
P. O. BOX 3267
HOUSTON, TX 77253-3267 (US)

(21) Appl. No.: **09/961,463**

(22) Filed: **Sep. 24, 2001**

Related U.S. Application Data

(63) Continuation of application No. 09/206,677, filed on Dec. 7, 1998, now patented.

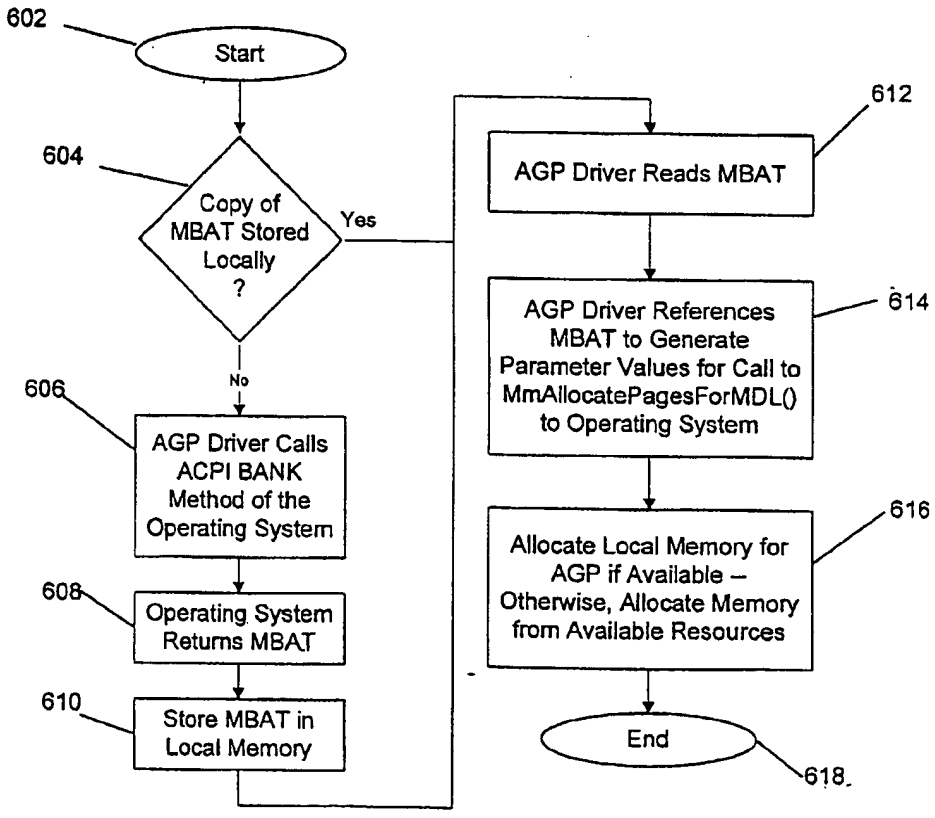
Publication Classification

(51) **Int. Cl.**[7] **G06F 12/02**

(52) **U.S. Cl.** **345/543**

(57) **ABSTRACT**

A computer system having a highly parallel system architecture with multiple central processing units, multiple core logic chipsets and pooled system memory is provided with one or more AGP ports capable of connection to AGP devices. A memory manager is provided within the operating system for allocating pooled memory resources without regard to the location of that memory. A method is presented for dynamically allocating memory for the AGP device that is located on the same core logic chipset to which the AGP device is connected. By allocating local memory instead of allocating memory on remote core logic units, the AGP device can access the needed memory quickly without memory transmissions along the host bus, thereby increasing overall performance of the computer system.



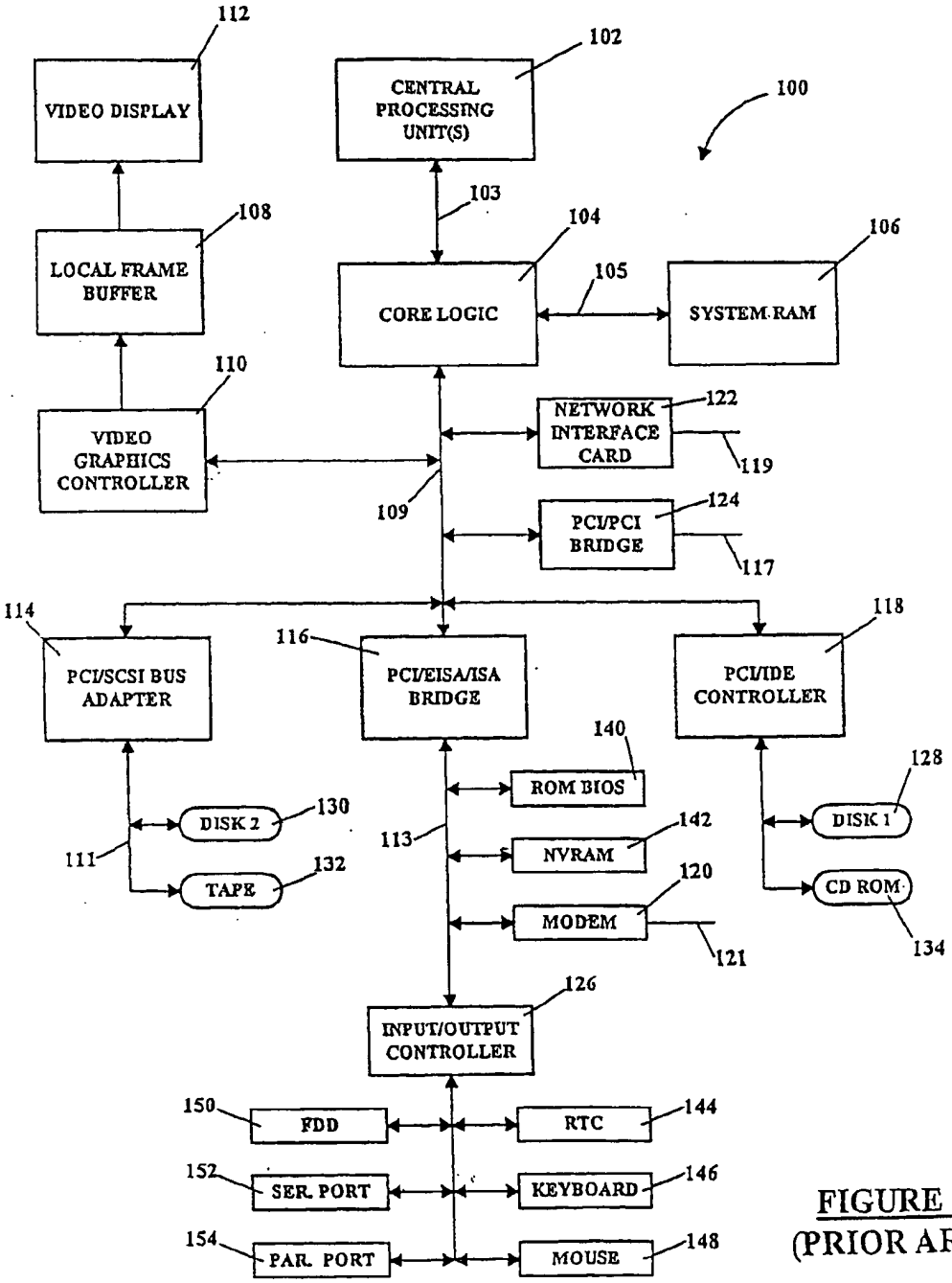


FIGURE 1
(PRIOR ART)

Figure 2
(prior art)

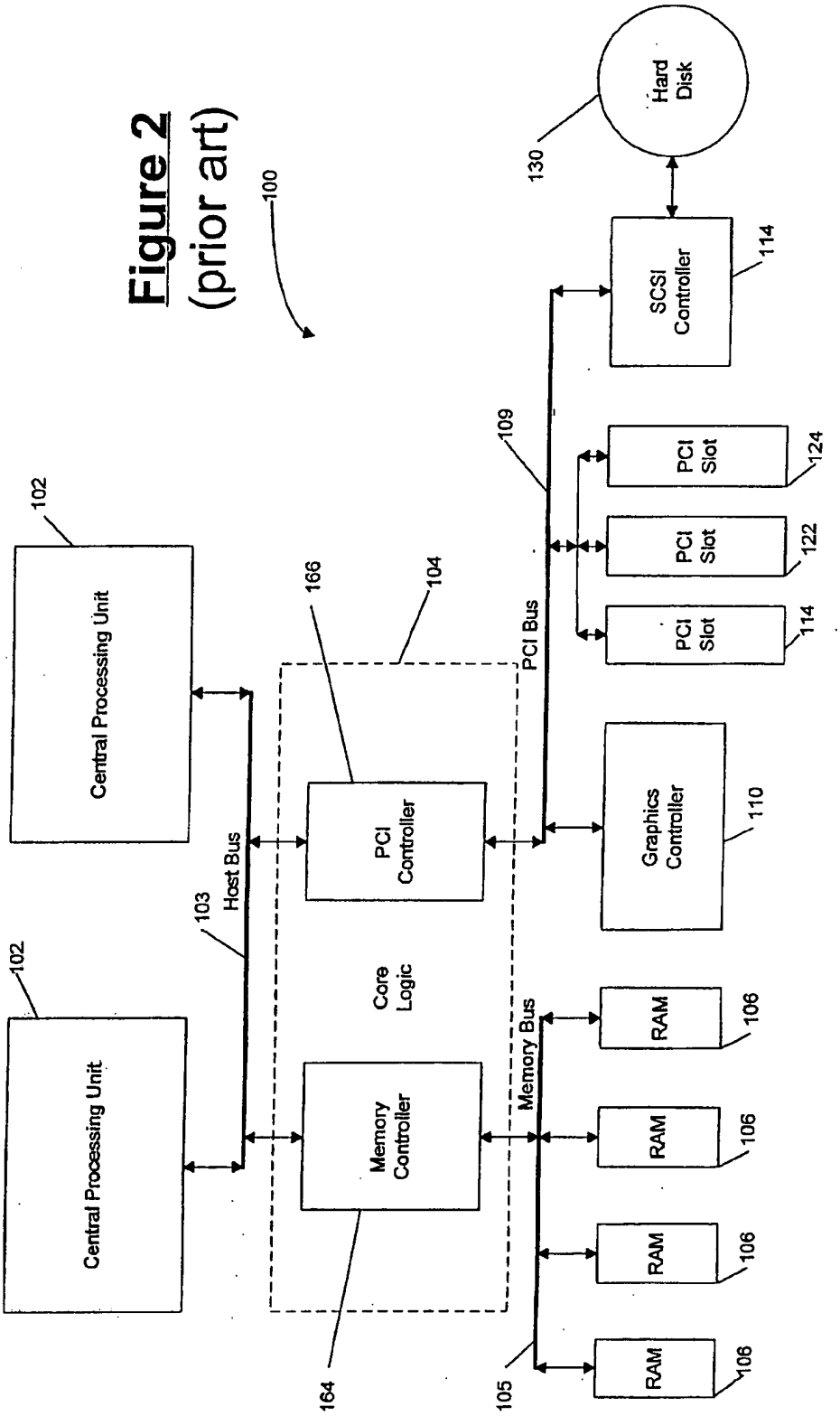


Figure 3

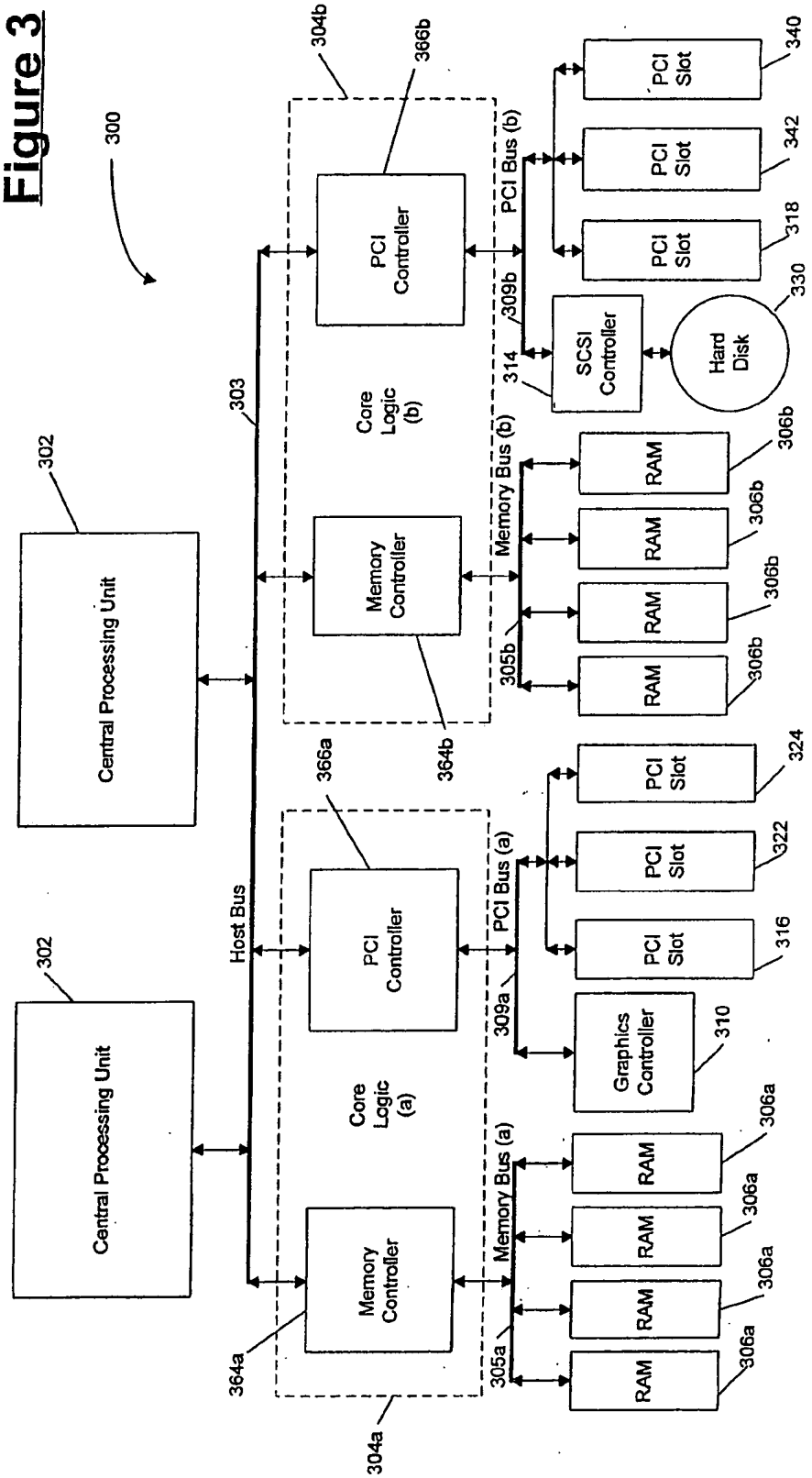
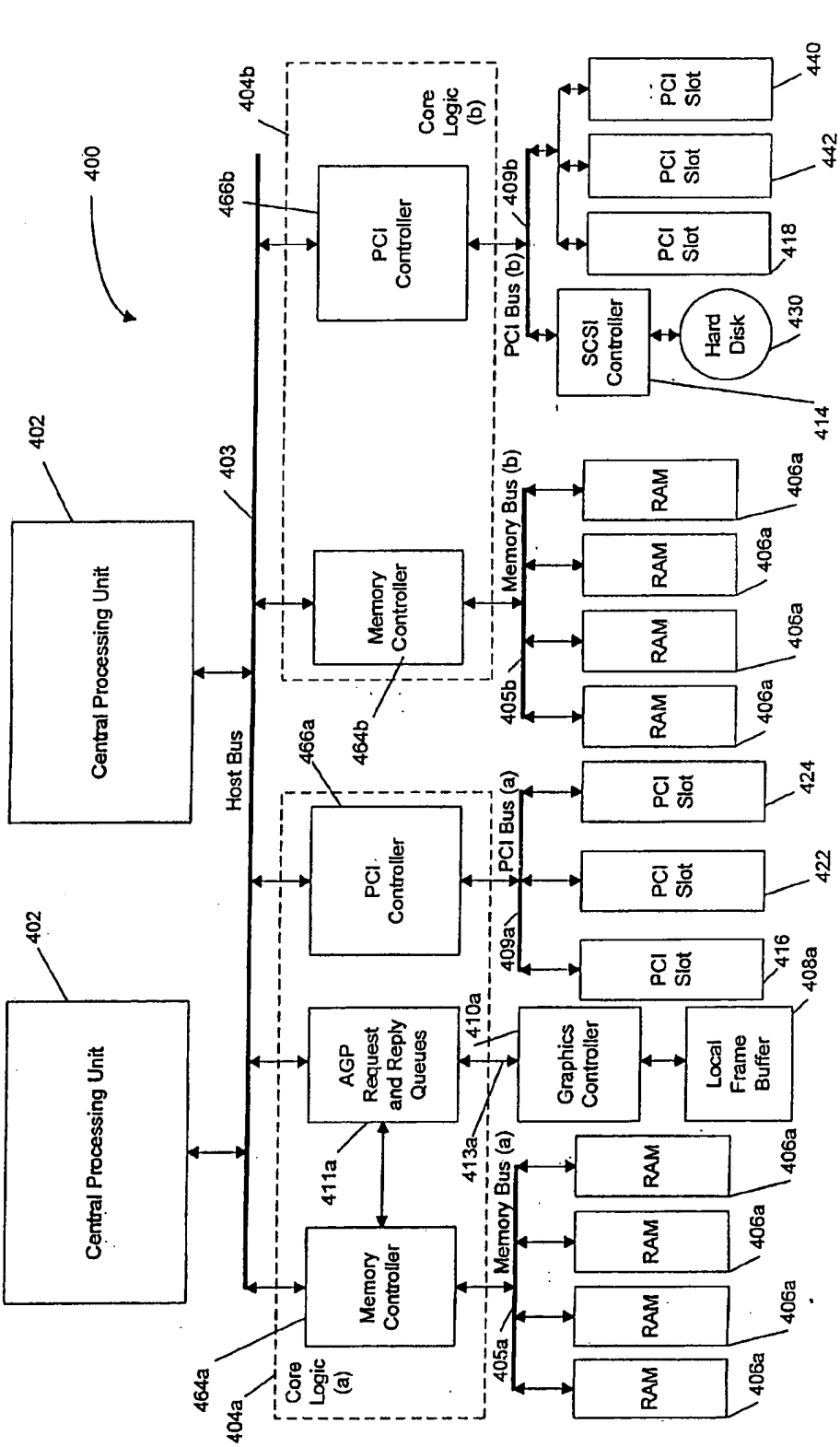

300
302
Central Processing Unit
302
Central Processing Unit
Host Bus
303
304a
Core Logic (a)
Memory Controller
364a
PCI Controller
366a
Memory Bus (a)
305a
RAM
306a
RAM
306a
RAM
306a
RAM
306a
PCI Bus (a)
309a
Graphics Controller
310
PCI Slot
316
PCI Slot
322
PCI Slot
324
304b
Core Logic (b)
Memory Controller
364b
PCI Controller
366b
Memory Bus (b)
305b
RAM
306b
RAM
306b
RAM
306b
RAM
306b
PCI Bus (b)
309b
SCSI Controller
314
Hard Disk
330
PCI Slot
318
PCI Slot
342
PCI Slot
340

Figure 4


400
Central Processing Unit
402
Central Processing Unit
402
Host Bus
403
Core Logic (a)
404a
Core Logic (b)
404b
PCI Controller
466b
PCI Controller
466a
Memory Controller
464b
Memory Controller
464a
AGP Request and Reply Queues
411a
Graphics Controller
410a
Local Frame Buffer
408a
PCI Bus (b)
409b
PCI Bus (a)
409a
Memory Bus (b)
405b
Memory Bus (a)
405a
SCSI Controller
414
Hard Disk
430
PCI Slot
440
PCI Slot
442
PCI Slot
418
PCI Slot
424
PCI Slot
422
PCI Slot
416
RAM
406a
RAM
406a
RAM
406a
RAM
406a
RAM
406a
RAM
406a
RAM
406a
RAM
406a

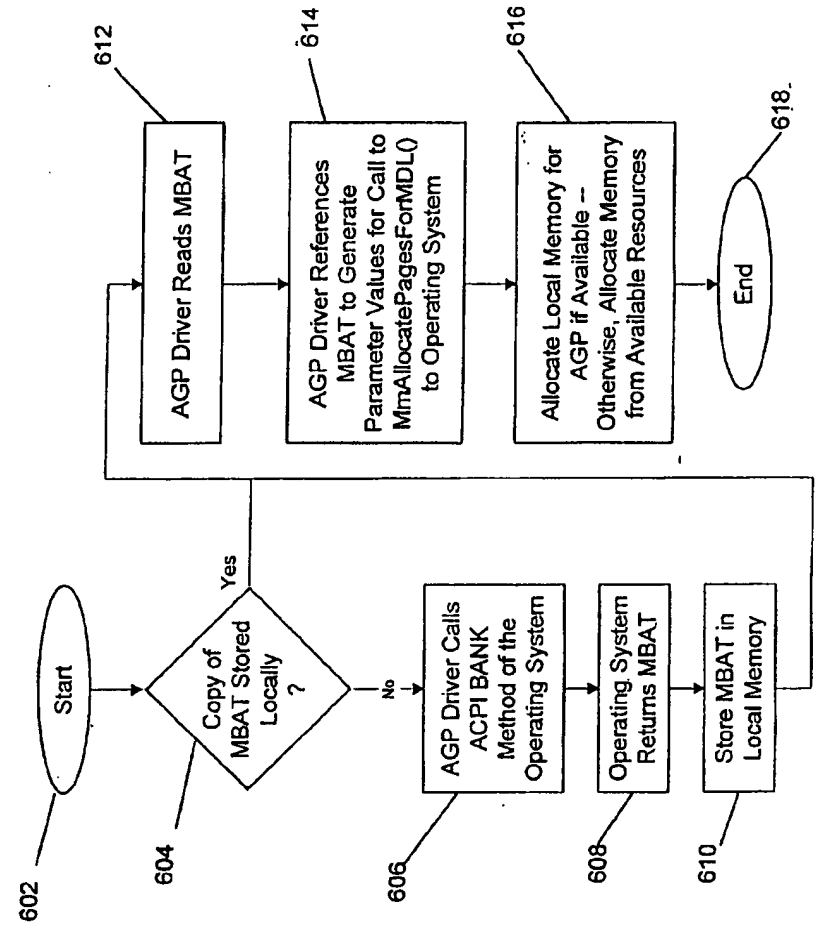


Figure 6

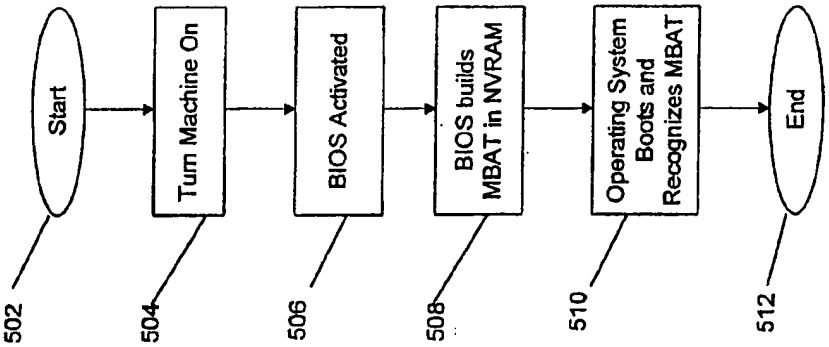


Figure 5

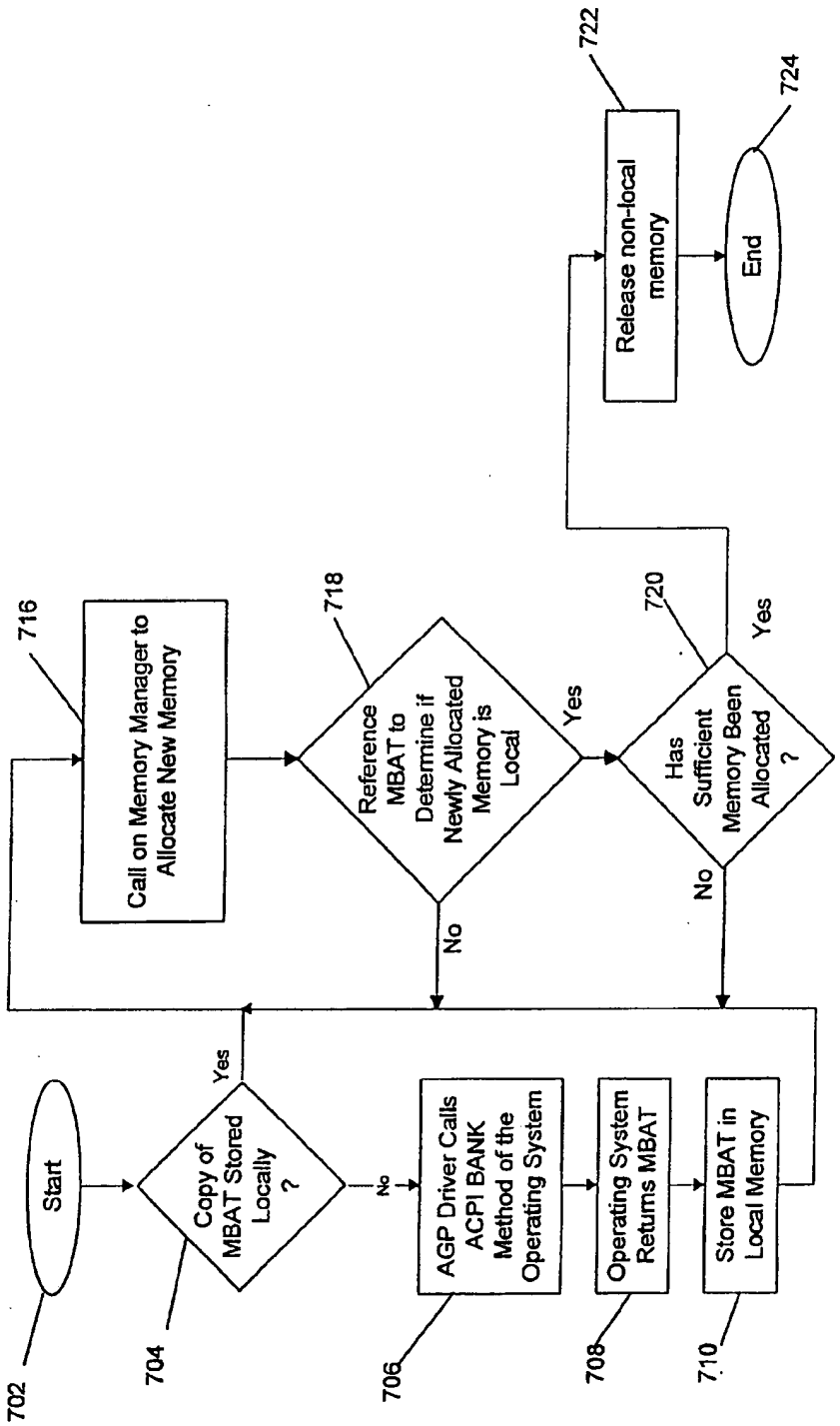
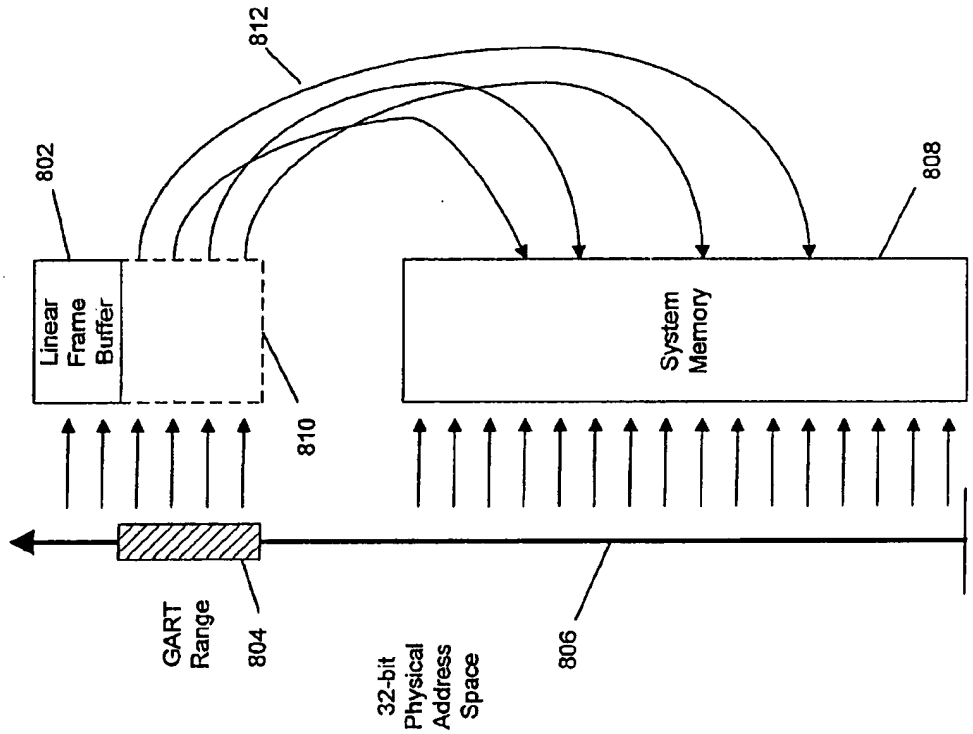

Start
Copy of MBAT Stored Locally ?
Yes
No
AGP Driver Calls ACPI BANK Method of the Operating System
Operating System Returns MBAT
Store MBAT in Local Memory
Call on Memory Manager to Allocate New Memory
Reference MBAT to Determine if Newly Allocated Memory is Local
Yes
No
Has Sufficient Memory Been Allocated ?
Yes
No
Release non-local memory
End
702
704
706
708
710
716
718
720
722
724

Figure 7

Figure 8


812
802
808
Linear Frame Buffer
System Memory
810
GART Range
804
32-bit Physical Address Space
806

METHOD AND SYSTEM FOR ALLOCATING MEMORY FROM THE LOCAL MEMORY CONTROLLER IN A HIGHLY PARALLEL SYSTEM ARCHITECTURE (HPSA)

CROSS REFERENCE TO RELATED APPLICATION

[0001] This patent application is related to commonly owned U.S. patent application Ser. No. 07/926,422, filed on Sep. 9, 1997, entitled "SYSTEM AND METHOD FOR DYNAMICALLY ALOCATING ACCELERATED GRAPHICS PORT MEMORY SPACE" by Ronald T. Horan, Phillip M. Jones, Gregory N. Santos, Robert Allan Lester, and Robert C. Elliott; and Ser. No. 08/925,722, filed on Sep. 9, 1997, entitled "GRAPHICS ADDRESS REMAPPING TABLE ENTRY FEATURE FLAGS FOR CUSTOMIZING THE OPERATION OF MEMORY PAGES WITH AN ACCELERATED GRAPHICS PORT DEVICE" by Ronald T. Horan, Phillip M. Jones, Gregory N. Santos, Robert Allan Lester, and Robert C. Elliott, and are hereby incorporated by reference for all purposes.

BACKGROUND OF THE INVENTION

[0002] 1. Field of the Invention

[0003] The present invention relates to computer systems using at least one accelerated graphics port (AGP) with at least two core logic chip sets, and more particularly, in allocating AGP/GART memory from the system memory local to the AGP device.

[0004] 2. Description of the Related Technology

[0005] Use of computers, especially personal computers, in business and at home is becoming more and more pervasive because the computer has become an integral tool of most information workers who work in the fields of accounting, law, engineering, insurance, services, sales and the like. Rapid technological improvements in the field of computers have opened up many new applications heretofore unavailable or too expensive for the use of older technology mainframe computers. These personal computers may be stand-alone workstations (high-end individual personal computers), desktop personal computers, portable laptop computers and the like. Moreover, personal computers may be linked together in a network by a "network server" which is also a personal computer which may have a few additional features specific to its purpose in the network. The network server may be used to store massive amounts of data, and may facilitate interaction of the individual workstations connected to the network for electronic mail ("E-mail"), document databases, video teleconferencing, white boarding, integrated enterprise calendar, virtual engineering design and the like. Multiple network servers may also be interconnected by local area networks ("LAN") and wide area networks ("WAN").

[0006] A significant part of the ever-increasing popularity of the personal computer, besides its low cost relative to just a few years ago, is its ability to run sophisticated programs and perform many useful and new tasks. Personal computers today may be easily upgraded with new peripheral devices for added flexibility and enhanced performance. A major advance in the performance of personal computers (both workstation and network servers) has been the implementation of sophisticated peripheral devices such as video graphics adapters, local area network interfaces, SCSI bus adapters, full motion video, redundant error checking and correcting disk arrays, and the like. These sophisticated peripheral devices are capable of data transfer rates approaching the native speed of the computer system's microprocessor central processing unit ("CPU"). The peripheral devices' data transfer speeds are achieved by connecting the peripheral devices to the microprocessor(s) and associated system random access memory through high-speed expansion local buses. Most notably, a high speed expansion local bus standard has emerged that is microprocessor independent and has been embraced by a significant number of peripheral hardware manufacturers and software programmers. This high-speed expansion bus standard is called the "Peripheral Component Interconnect" or "PCI." A more complete definition of the PCI local bus may be found in the following specifications: PCI Local Bus Specification, revision 2.1; PCI/PCI Bridge Specification, revision 1.0; PCI System Design Guide, revision 1.0; PCI BIOS Specification, revision 2.1, and Engineering Change Notice ("ECN") entitled "Addition of 'New Capabilities' Structure," dated May 20, 1996, the disclosures of which are hereby incorporated by reference. These PCI specifications and ECN's are available from the PCI Special Interest Group, P.O. Box 14070, Portland, Oreg. 97214.

[0007] A computer system has a plurality of information (data and address) busses. These busses include a host bus, a memory bus, at least one high-speed expansion local bus such as the PCI bus, and other peripheral buses such as the Small Computer System Interface (SCSI), Extension to Industry Standard Architecture (EISA), and Industry Standard Architecture (ISA). The microprocessor(s) of the computer system communicates with main memory and with the peripherals that make up the computer system over these various buses. The microprocessor(s) communicates to the main memory over a host bus to a memory bus bridge. The peripherals, depending on their data transfer speed requirements, are connected to the various buses which are connected to the microprocessor host bus through bus bridges that detect required actions, arbitrate, and translate both data and addresses between the various buses.

[0008] Increasingly sophisticated microprocessors have revolutionized the role of the personal computer by enabling complex applications software to run at mainframe computer speeds. The latest microprocessors have brought the level of technical sophistication to personal computers that, just a few years ago, was available only in mainframe and mini-computer systems. Some representative examples of these new microprocessors are the "PENTIUM" and "PENTIUM PRO" (registered trademarks of Intel Corporation). Advanced Micro Devices, Cyrix, IBM, Digital Equipment Corp., and Motorola also manufacture advanced microprocessors.

[0009] These sophisticated microprocessors have, in turn, made possible running complex application programs using advanced three dimensional ("3-D") graphics for computer aided drafting and manufacturing, engineering simulations, games and the like. Increasingly complex 3-D graphics require higher speed access to ever-larger amounts of graphics data stored in memory. This memory may be part of the video graphics processor system, but, preferably, would be best (lowest cost) if part of the main computer system

memory. Intel Corporation has proposed a low cost but improved 3-D graphics standard called the "Accelerated Graphics Port" (AGP) initiative. With AGP 3-D, graphics data, in particular textures, may be shifted out of the graphics controller local memory to computer system memory. The computer system memory is lower in cost than the graphics controller local memory and is more easily adapted for a multitude of other uses besides storing graphics data.

[0010] The Intel AGP 3-D graphics standard defines a high-speed data pipeline, or "AGP bus," between the graphics controller and system memory. This AGP bus has sufficient bandwidth for the graphics controller to retrieve textures from system memory without materially affecting computer system performance for other non-graphics operations. The Intel 3-D graphics standard is a specification, which provides signal, protocol, electrical, and mechanical specifications for the AGP bus and devices attached thereto. The original specification is entitled "Accelerated Graphics Port Interface Specification Revision 1.0," dated Jul. 31, 1996, the disclosure of which is hereby incorporated by reference. The newest specification is entitled "Preliminary Draft of Accelerated Graphics Port Interface Specification, Preliminary Draft of Revision 2.0," dated Dec. 10, 1997. These AGP Specifications are available from Intel Corporation, Santa Clara, Calif.

[0011] The AGP interface specification uses the 66 MHz PCI (Revision 2.1) specification as an operational baseline, with three performance enhancements to the PCI specification which are used to optimize the AGP Specification for high performance 3-D graphics applications. These enhancements are: 1) pipelined memory read and write operations, 2) demultiplexing of address and data on the AGP bus by use of side-band signals, and 3) data transfer rates of 133 MHz for data throughput in excess of 500 megabytes per second ("MB/s"). The remaining AGP Specification does not modify the PCI specification, but rather provides a range of graphics-oriented performance enhancements for use by 3-D graphics hardware and software designers. The AGP Specification is neither meant to replace or diminish full use of the PCI standard in the computer system. The AGP Specification creates an independent and additional high speed local bus for use by 3-D graphics devices such as a graphics controller, wherein the other input-output ("I/O") devices of the computer system may remain on any combination of the PCI, SCSI, EISA and ISA buses.

[0012] To functionally enable this AGP 3-D graphics bus, new computer system hardware and software are required. This requires new computer system core logic designed to function as a host bus/memory bus/PCI bus to AGP bus bridge meeting the AGP Specification. Moreover, new Read Only Memory Basic Input Output System ("ROM BIOS") and Application Programming Interface ("API") software are required to make the AGP dependent hardware functional in the computer system. The computer system core logic must still meet the PCI standards referenced above and facilitate interfacing the PCI bus(es) to the remainder of the computer system. In addition, new AGP compatible device cards must be designed to properly interface, mechanically and electrically, with the AGP bus connector.

[0013] AGP and PCI device cards are neither physically nor electrically interchangeable even though there is some commonality of signal functions between the AGP and PCI interface specifications. The present AGP Specification only makes allowance for a single AGP device on an AGP bus, whereas, the PCI specification allows two plug-in slots for PCI devices plus a bridge on a PCI bus running at 66 MHz. The single AGP device is capable of functioning in both a 1× mode (264 MB/s peak) and a 2× mode (532 MB/s peak). The AGP bus is defined as a 32-bit bus, and may have up to four bytes of data transferred per clock in the 1× mode and up to eight bytes of data per clock in the 2× mode. The PCI bus is defined as either a 32 bit or 64 bit bus, and may have up to four or eight bytes of data transferred per clock, respectively. The AGP bus, however, has additional side-band signals which enables it to transfer blocks of data more efficiently than is possible using a PCI bus. An AGP bus running in the 2× mode provides sufficient video data throughput (532 MB/s peak) to allow increasingly complex 3-D graphics applications to run on personal computers.

[0014] A major performance/cost enhancement using AGP in a computer system is accomplished by shifting texture data structures from local graphics memory to main memory. Textures are ideally suited for this shift for several reasons. Textures are generally read-only, and therefore problems of access ordering and coherency are less likely to occur. Shifting of textures serves to balance the bandwidth load between system memory and the local graphics memory because a well-cached host processor has much lower memory bandwidth requirements than does a 3-D rendering machine. Texture access comprises perhaps the single largest component of rendering memory bandwidth, so avoiding loading or caching textures in local graphics memory saves not only this component of local memory bandwidth, but also the bandwidth necessary to load the texture store in the first place. Furthermore, texture data must pass through main memory anyway as it is loaded from a mass store device. Texture size is dependent upon application quality rather than on display resolution, and therefore may require the greatest increase in memory as software applications become more advanced. Texture data is not persistent and may reside in the computer system memory only for the duration of the software application. Consequently, any system memory spent on texture storage can be returned to the free memory heap when the application concludes (unlike a graphic controller's local frame buffer, which may remain in persistent use). For these reasons, shifting texture data from local graphics memory to main memory significantly reduces computer system costs when implementing 3-D graphics.

[0015] Generally, in computer system memory architecture, the graphics controller's physical address space resides above the top of system memory. The graphics controller uses this physical address space to access its local memory, which holds information that is required to generate a graphics screen. In the AGP system, information still resides in the graphics controller's local memory (textures, alpha, z-buffer, etc.), but some data which previously resided in this local memory is moved to system memory (primarily textures, but also command lists, etc.). The address space employed by the graphics controller to access these textures becomes virtual, meaning that the physical memory corresponding to this address space doesn't actually exist above the top of the memory space. In reality, each of these virtual addresses corresponds to a physical address in system memory. The graphics controller sees this virtual address

space, referenced hereinafter as "AGP device address space," as one contiguous block of memory, but the corresponding physical memory addresses may be allocated in 4 kilobyte ("KB"), non-contiguous pages throughout the computer system physical memory.

[0016] There are two primary AGP usage models for 3D rendering that have to do with how data are partitioned and accessed, and the resultant interface data flow characteristics. In the "DMA" model, the primary graphics memory is a local memory referred to as 'local frame buffer' and is associated with the AGP graphics controller or "video accelerator." 3D structures are stored in system memory, but are not used (or "executed") directly from this memory; rather they are copied to primary (local) memory, to which the rendering engine's address generator (of the AGP graphics controller) makes references thereto. This implies that the traffic on the AGP bus tends to be long, sequential transfers, serving the purpose of bulk data transport from system memory to primary graphics (local) memory. This sort of access model is amenable to a linked list of physical addresses provided by software (similar to operation of a disk or network I/O device), and is generally not sensitive to a non-contiguous view of the memory space.

[0017] In the "execute" model, the video accelerator uses both the local memory and the system memory as primary graphics memory. From the accelerator's perspective, the two memory systems are logically equivalent; any data structure may be allocated in either memory, with performance optimization as the only criteria for selection. In general, structures in system memory space are not copied into the local memory prior to use by the video accelerator, but are "executed" in place. This implies that the traffic on the AGP bus tends to be short, random accesses, which are not amenable to an access model based on software resolved lists of physical addresses. Since the accelerator generates direct references into system memory, a contiguous view of that space is essential. But, since system memory is dynamically allocated in, for example, random 4,096 byte blocks of the memory, hereinafter 4-kilobyte ("KB") pages, it is necessary in the "execute" model to provide an address mapping mechanism that maps the random 4 KB pages into a single contiguous address space.

[0018] The AGP Specification supports both the "DMA" and "execute" models. However, since a primary motivation of the AGP is to reduce growth pressure on the graphics controller's local memory (including local frame buffer memory), the "execute" model is preferred. Consistent with this preference, the AGP Specification requires a virtual-to-physical address re-mapping mechanism, which ensures the graphics accelerator (AGP master) will have a contiguous view of graphics data structures dynamically allocated in the system memory. This address re-mapping applies only to a single, programmable range of the system physical address space and is common to all system agents. Addresses falling in this range are re-mapped to non-contiguous pages of physical system memory. All addresses not in this range are passed through without modification, and map directly to main system memory, or to device specific ranges, such as a PCI device's physical memory. Re-mapping is accomplished via a "Graphics Address Remapping Table" ("GART") which is set up and maintained by a GART miniport driver software, and used by the core logic chipset to perform the re-mapping. In order to avoid compatibility issues and allow future implementation flexibility, this mechanism is specified at a software (API) level. In other words, the actual GART format may be abstracted to the API by a hardware abstraction layer ("HAL") or mini-port driver that is provided with the core logic chipset. While this API does not constrain the future partitioning of re-mapping hardware, the re-mapping function will typically be implemented in the core logic chipset.

[0019] The contiguous AGP graphics controller's device addresses are mapped (translated) into corresponding physical addresses that reside in the computer system physical memory by using the GART which may also reside in physical memory. The GART is used by the core logic chipset to re-map AGP device addresses that can originate from either the AGP, host, or PCI buses. A software program called a "GART miniport driver" manages the GART. The GART miniport driver provides GART services for the computer software operating system.

[0020] The size of AGP device address space is always greater than or equal to the amount of physical system memory allocated to AGP. The amount of physical memory allocated to AGP is managed by the computer operating system. AGP device address space is specific to the core logic chipset. In the Compaq/RCC chipset, the AGP device address space may be from a minimum of 32 MB to a maximum of 2 GB, with a default value of, for example, 256 MB. In the Intel 440 BX chipset, the minimum is 4 MB and the maximum is 256 MB. Some AGP graphics controllers do not require the default value of 256 MB of device address space, and may only need the minimum of 32 MB.

[0021] AGP memory issues are complicated greatly in computer systems that utilize multiple central processing units and multiple core logic chipsets to facilitate parallel processing. These memory issues are compounded if there are multiple AGP busses present in the computer system. For example, AGP or GART physical system memory can be allocated to memory residing on the non-AGP memory controller controlled through another chipset. Although these addresses can be passed between controllers, doing so consumes both valuable bandwidth and time.

[0022] What is needed is a system and method of dynamically allocating virtual address space for an AGP device in a multi-processor/multi-core logic computer system without having to allocate memory in a portion of the computer system that would minimize communications between different memory controllers on separate core logic units.

SUMMARY OF THE INVENTION

[0023] The present invention solves the problems inherent in the prior art systems by providing an advanced configuration and power interface (ACPI) control method. ACPI is an operating-system-based specification that defines a flexible and abstract hardware interface for desktop and notebook PCs. As such, the specification enables system designers to integrate power management throughout the hardware components, the operating system, and the software applications. ACPI is utilized in both the Windows NT™ and the Windows 98™ operating systems, available from the Microsoft Corporation of Redmond, Wash. The method of the present invention calls for creating a new ACPI table called the Memory Bank Address Table (MBAT). System BIOS utilizes ACPI Source Language (ASL) to build the

MBAT dynamically at startup. The AGP driver and the GART miniport driver access the MBAT via an ACPI control method. The AGP driver and the GART miniport driver uses the MBAT information when making calls to the operating system's memory manager to allocate new memory. For example, the drivers pass ranges derived from MBAT information to the memory manager when the Windows NT 5.0™ operating system service routine "MmAllocatePagesForMdl( )" is called. The memory manager attempts to allocate local memory within the specified range. If local memory cannot be allocated within the required range, the drivers allocate memory from outside the range as a fall back. Once the memory is allocated, the AGP driver and the GART miniport driver access the allocated memory in the normal manner. If local memory is allocated, then the AGP device can utilize this memory without the need of the host bus, thereby eliminating the host bus-related latencies as well as conserving host bus bandwidth.

[0024] The MBAT is built before the need for memory allocation arises. In the preferred embodiment of the present invention, the MBAT is built dynamically by the system BIOS as needed. However, if the system configuration never changes, the MBAT may be built once at system startup and simply stored until needed. If the system BIOS does not support the building of the MBAT, the GART miniport driver can build the MBAT. The structure of the MBAT contains the ranges of memory local to the AGP device depending upon the configuration of the computer system found during startup (POST). The actual form of the MBAT is chipset specific because of the use of certain registers. In any case, by the time that the operating system has booted, the MBAT will have been created and the operating system, specifically the memory manager of the operating system, will know of the location of the MBAT. The location of the MBAT can be in system memory, in non-volatile RAM (NVRAM), on a hard disk drive, a network drive that is connected to the computer system via a network interface card, or on any volatile or non-volatile memory that can be connected to the computer system which can contain the contents of the MBAT.

[0025] In operation, when the GART miniport driver or the AGP driver need to allocate memory, they reference the MBAT first. In some cases, such as when the MBAT resides on non-local system memory, the AGP driver will receive a copy of the MBAT for future reference. Referencing the copy of the MBAT locally by the AGP driver further reduces demands upon the host bus of the computer system. Once the MBAT has been referenced, either the AGP driver or the GART miniport driver will use the ranges of the system memory that reside on the same core logic chipset as the AGP bus. It is these ranges that are used to allocate memory for the AGP driver or GART miniport driver. If the range of local memory is unavailable (i.e., it has already been allocated for other devices), then the AGP driver or GART miniport driver request memory from any available resources as a fall back.

[0026] Other and further features and advantages will be apparent from the following description of presently preferred embodiments of the invention, given for the purpose of disclosure and taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0027] **FIG. 1** is a schematic block diagram of a prior art computer system;

[0028] **FIG. 2** is a schematic block diagram of a computer system according to the present invention;

[0029] **FIG. 3** is a schematic functional block diagram of an embodiment of the present invention according to the computer system of **FIG. 2**;

[0030] **FIG. 4** is a flowchart of the method of to build the MBAT of the present invention;

[0031] **FIG. 5** is a flowchart of the building of the MBAT of the present invention;

[0032] **FIG. 6** is a flowchart of the allocation of memory according to the present invention;

[0033] **FIG. 7** is a flowchart of the allocation of memory according to an alternate embodiment of the present invention; and

[0034] **FIG. 8** is a schematic diagram of memory allocation according to the AGP specification.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

[0035] The present invention is an apparatus, method and system for allocating AGP/GART local memory whenever possible from the local memory controller to which the AGP device is connected.

[0036] The present invention provides a core logic chipset in a computer system which is capable of implementing a bridge between host processor and memory buses, an AGP bus adapted for an AGP device(s), and a PCI bus adapted for PCI devices. The AGP device may be a graphics controller which utilizes graphical data, such as textures, by addressing a contiguous virtual address space, hereinafter "AGP device address space," that is translated from non-contiguous memory pages located in the computer system physical memory by the core logic chipset. The core logic chipset utilizes a "Graphics Address Remapping Table" ("GART") which may reside in a physical memory of the computer system, such as system random access memory, and may be controlled by the core logic chipset's software driver(s). The function of the GART is to re-map virtual addresses referenced by the AGP device to the physical addresses of the graphics information located in the computer system physical memory (see **FIG. 8**). Each entry of the GART describes a first byte address location for a page of physical memory. The page of physical memory may be 4,096 bytes (4 KB) in size. A GART entry comprises a memory address translation pointer and software controllable feature flags (see the AGP Specification). These feature flags may be used to customize the associated page of physical memory. API software and miniport drivers may write to and/or read from these feature flags.

[0037] For illustrative purposes, preferred embodiments of the present invention are described hereinafter for computer systems utilizing the Intel x86 microprocessor architecture and certain terms and references will be specific to those processor platforms. AGP and PCI are interface standards are, however, hardware independent and may be utilized with any host computer designed for these interface

standards. It will be appreciated by those skilled in the art of computer systems that the present invention may be adapted and applied to any computer platform utilizing the AGP and PCI interface standards.

[0038] The PCI specifications referenced above are readily available and are hereby incorporated by reference. The AGP Specification entitled "Accelerated Graphics Port Interface Specification, Revision 1.0," dated Jul. 31, 1996, as referenced above, is readily available from Intel Corporation, and is hereby incorporated by reference. Further definition and enhancement of the AGP Specification is more fully defined in "Compaq's Supplement to the 'Accelerated Graphics Port Interface Specification Version 1.0'," Revision 0.8, dated Apr. 1, 1997, and is hereby incorporated by reference. Both of these AGP specifications were included as Appendices A and B in commonly owned co-pending U.S. patent application Ser. No. 08/853,289; filed May 9, 1997, entitled "Dual Purpose Apparatus, Method and System for Accelerated Graphics Port and Peripheral Component Interconnect" by Ronald T. Horan and Sompong Olarig, and which is hereby incorporated by reference.

[0039] Referring now to the drawings, the details of preferred embodiments of the present invention are schematically illustrated. Like elements in the drawings will be represented by like numbers, and similar elements will be represented by like numbers with a different lower case letter suffix. Referring now to **FIG. 1, a** schematic block diagram of a computer system illustrates the basic architecture of prior art systems. A computer system is generally indicated by the numeral **100** and comprises a central processing unit(s) ("CPU") **102**, core logic chipset **104**, system random access memory ("RAM") **106**, a video graphics controller **110**, a local frame buffer **108**, a video display **112**, a PCI/SCSI bus adapter **114**, a PCI/EISA/ISA bridge **116**, and a PCI/IDE controller **118**. Single or multilevel cache memory (not illustrated) may also be included in the computer system **100** according to the current art of microprocessor computer systems. The CPU **102** may be a plurality of CPUs **102** in a symmetric or asymmetric multi-processor configuration.

[0040] The CPU(s) **102** is connected to the core logic chipset **104** through a host bus **103**. The system RAM **106** is connected to the core logic chipset **104** through. a memory bus **105**. The video graphics controller(s) **110** is connected to the core logic chipset **104** through a primary PCI bus **109**. The PCI/SCSI bus adapter **114**, PCI/EISA/ISA bridge **116**, and PCI/IDE controller **118** are connected to the core logic chipset **104** through the primary PCI bus **109**. Also connected to the PCI bus **109** are a network interface card ("NIC") **122** and a PCI/PCI bridge **124**. Some of the PCI devices such as the NIC **122** and PCI/PCI bridge **124** may plug into PCI connectors on the computer system **100** motherboard (not illustrated).

[0041] Hard disk **130** and tape drive **132** are connected to the PCI/SCSI bus adapter **114** through a SCSI bus **111**. The NIC **122** is connected to a local area network **119**. The PCI/EISA/ISA bridge **116** connects over an EISA/ISA bus **113** to a ROM BIOS **140**, non-volatile random access memory (NVRAM) **142**, modem **120**, and input-output controller **126**. The modem **120** connects to a telephone line **121**. The input-output controller **126** interfaces with a keyboard **146**, real time clock (RTC) **144**, mouse **148**, floppy disk drive ("FDD") **150**, and serial/parallel ports **152, 154**. The EISA/ISA bus **113** is a slower information bus than the PCI bus **109**, but it costs less to interface with the EISA/ISA bus **113**.

[0042] **FIG. 2** is a simplified schematic block diagram of the computer system of **FIG. 1**. Identical components are numbered identically between the two figures. **FIG. 2**, however, shows more details of the core logic unit **104**. Specifically, the PCI controller **166** and the Memory controller **164**, and their connections to the memory bus **105** and the PCI bus **109**, respectively, are illustrated.

[0043] Traditionally, PC systems have contained only one root I/O bridge and one memory controller (see **FIG. 2**). In certain high-bandwidth applications found in workstations and servers, these systems can bottleneck at either the PCI controller **166** or the memory controller **164**. To avoid these limitations, system designers are creating highly parallel architectures incorporating multiple memory controllers and multiple core logic units.

[0044] **FIG. 3** shows a computer system **300** having two core logic units **304***a* and **304***b*. As shown in **FIG. 3**, system RAM **306***a* is connected to a memory bus **305***a*, which, in turn, is connected to the first memory controller **364***a* of the first core logic unit **304***a*. PCI slots **316, 322,** and **324** are connected as a unit to the first PCI bus **309***a*. A graphics controller **310** is also connected to the PCI bus **309***a*. The PCI bus **309***a* is connected to the PCI controller **366***a* of the first core logic unit **304***a*. The second core logic unit **304***b* contains its own bank of system memory **306***b* which, as with the system memory **306***a*, contributes to the overall memory resources of the computer system **300**. As with the first core logic unit, the system memory **306***b* of the second core logic unit **304***b* is connected to a second memory bus **305***b* which, in turn, is connected to the second memory controller **364***b*. Similarly, the PCI slots **318, 342** and **340** are connected to the second PCI bus **309***b*. A hard disk drive **330** is also connected to the second PCI bus **309***b* via SCSI controller **314**. A second PCI controller **366***b* links the second PCI bus **309***b* to the host bus **303**. Both of the core logic units **304***a* and **304***b* are connected to the host bus **303** as shown in **FIG. 3**. It is through the host bus **303** that data in memory from the second bank of system RAM **306***b* is transferred to, for example, graphics controller **310** of the first core logic unit **304***a*. Although these enhanced memory and I/O architectures do not directly affect operating systems, specifically memory managers, one class of devices is affected—AGP.

[0045] **FIG. 4** shows the preferred embodiment of a computer system **400** of the present invention. As **FIG. 4** shows, the computer system **400** has two core logic units **404***a* and **404***b* in a highly parallel system architecture (although more CPUs can be added). As shown in **FIG. 4**, system RAM **406***a* is connected to a memory bus **405***a*, which, in turn, is connected to the first memory controller **464***a* of the first core logic unit **404***a*. PCI slots **416, 422,** and **424** are connected as a unit to the first PCI bus **409***a*. A graphics controller **410** is connected to the AGP bus **413**, which, in turn, is connected to the AGP request and reply queues **411** that link the AGP bus **413** to the host bus **403** and the first memory controller **464***a*. The PCI bus **409***a* is connected to the PCI controller **466***a* of the first core logic

unit **404***a*. The second core logic unit **404***b* contains its own bank of system memory **406***b* which, as with the system memory **406***a*, contributes to the overall memory resources of the computer system **400**. As with the first core logic unit, the system memory **406***b* of the second core logic unit **404***b* is connected to a second memory bus **405***b* which, in turn, is connected to the second memory controller **464***b*. Similarly, the PCI slots **418**, **442** and **440** are connected to the second PCI bus **409***b*. A hard disk drive **430** is also connected to the second PCI bus **409***b* via SCSI controller **414**. A second PCI controller **466***b* links the second PCI bus **409***b* to the host bus **403**. Both of the core logic units **404***a* and **404***b* are connected to the host bus **403** as shown in **FIG. 4**. It is through the host bus **403** that data in memory from the second bank of system RAM **406***b* is transferred to, for example, graphics controller **410** of the first core logic unit **404***a*. According to the highly parallel system architecture, system resources share all of the system memory **406***a* and **406***b*, regardless of the location of the specific device within the computer system **400**. When the operating system allocates memory, it does so as viewing all of the system memory as one contiguous unit.

[0046] AGP Specification

[0047] The Intel AGP Specification entitled "Preliminary Draft of Accelerated Graphics Port Interface, Preliminary Draft Specification Revision 2.0," dated Dec. 10, 1997, incorporated by reference hereinabove, provides signal, protocol, electrical, and mechanical specifications for the AGP bus. However, further design must be implemented before a fully functional computer system with AGP capabilities is realized. The following disclosure defines the implementation specific parts of an AGP interface according to the present invention. The following disclosure includes the GART and related technologies so that one of ordinary skill in the art may practice the present invention without undue experimentation when used with the aforementioned Intel AGP Specification incorporated by reference herein.

[0048] Moving textures and other information required by the graphics controller, such as command lists, out of the local frame buffer into system memory creates a problem: the presently implemented prior art computer system architecture, illustrated in **FIG. 1**, cannot support the bandwidth requirements of tomorrow's 3-D graphics enhanced applications. The standard PCI bus **109** (33 MHz, 32 bit) bandwidth is 132 MB/s peak and 50 MB/s typical. Microsoft Corporation estimates that future graphics applications will require in excess of 200 MB/s. This means that the PCI bus **109** in the computer system architecture illustrated in **FIG. 1** will likely starve the graphics controller **110** as well as other PCI devices (**122**, **124**, **114**, **116** and **118**) also trying to access the PCI bus **109**.

[0049] AGP Architecture

[0050] The AGP interface specification uses the 66 MHz PCI (PCI Local Bus Specification) specification as an operational baseline, and provides four significant performance extensions or enhancements to the PCI specification which are intended to optimize the A.G.P. for high performance 3D graphics applications. These A.G.P. extensions are not described in, or required by, the PCI Local Bus, Specification. These extensions are:

[0051] 1. Deeply pipelined memory read and write operations, fully hiding memory access latency.

[0052] 2. Demultiplexing of address and data on the bus, allowing almost 100% bus efficiency.

[0053] 3. New AC timing in the 3.3 V electrical specification that provides for one or two data transfers per 66-MHz clock cycle, allowing for real data throughput in excess of 500 MB/s.

[0054] 4. A new low voltage electrical specification that allows four data transfers per 66-MHz clock cycle, providing real data throughput of up to 1 GB/s.

[0055] These enhancements are realized through the use of "side-band" signals. The PCI specification has not been modified in any way, and the A.G.P. interface specification has specifically avoided the use of any of the "reserved" fields, encodings, pins, etc., in the PCI specification. The intent is to utilize the PCI design base while providing a range of graphics-oriented performance enhancements with varying complexity/performance tradeoffs available to the component provider.

[0056] AGP neither replaces nor diminishes the necessity of PCI in the system. This high-speed port (AGP) is physically, logically, and electrically independent of the PCI bus. It is an additional connection point in the system, as shown in **FIG. 4**. It is intended for the exclusive use of visual display devices with all other I/O devices remaining on the PCI bus. The add-in slot defined for AGP uses a new connector body (for electrical signaling reasons) which is not compatible with the PCI connector; PCI and AGP boards are not mechanically interchangeable.

[0057] Two Usage Models: "Execute" and "DMA"

[0058] There are two primary AGP usage models for 3D rendering that have to do with how data are partitioned and accessed, and the resultant interface data flow characteristics. In the "DMA" model, the primary graphics memory is the local memory associated with the accelerator, referred to as "local frame buffer." 3D structures are stored in system memory, but are not used (or "executed") directly from this memory; rather they are copied to primary (local) memory (the "DMA" operation) to which the rendering engine's address generator makes its references. This implies that the traffic on the AGP tends to be long, sequential transfers, serving the purpose of bulk data transport from system memory to primary graphics (local) memory. This sort of access model is amenable to a linked list of physical addresses provided by software (similar to operation of a disk or network I/O device), and is generally not sensitive to a non-contiguous view of the memory space.

[0059] In the "execute" model, the accelerator uses both the local memory and the system memory as primary graphics memory. From the accelerator's perspective, the two memory systems are logically equivalent; any data structure may be allocated in either memory, with performance optimization as the only criteria for selection. In general, structures in system memory space are not copied into the local memory prior to use by the accelerator, but are "executed" in place. This implies that the traffic on the AGP tends to be short, random accesses, which are not amenable to an access model based on software resolved lists of physical addresses. Because the accelerator generates direct references into system memory, a contiguous view of that space is essential. However, because system memory is

dynamically allocated in random 4K pages, it is necessary in the "execute" model to provide an address mapping mechanism that maps random 4K pages into a single contiguous, physical address space.

[0060] The AGP supports both the "DMA" and "execute" models. However, since a primary motivation of the A.G.P. is to reduce growth pressure on local memory, the "execute" model is the design center. Consistent with that emphasis, this interface specification requires a physical-to-physical address re-mapping mechanism which insures the graphics accelerator (an AGP master) will have a contiguous view of graphics data structures dynamically allocated in system memory.

[0061] This address re-mapping applies only to a single, programmable range of the system physical address space, as shown in **FIG. 8**. The 32-bit physical address space **806** shown is common to all system agents. Addresses falling in the GART range **804** are remapped **(812)** to non-contiguous pages of physical system memory **808**. All addresses not in this range are passed through without modification. From the processor's point of view, this means that requests are mapped directly to main system memory **808** or to device specific ranges, such as the graphics local frame buffer memory **802** shown in **FIG. 8**. From the AGP master's point of view, this means the request is only mapped to system memory **808**. Note that the core logic may optionally trap (not pass) requests out of the GART range **804** and report an error. If the mode of reporting the error is implemented, the core logic is required to default to the mode of not reporting an error after reset. How software enables the core logic to report an access out of the GART range as an error (if supported) is implementation specific.

[0062] Remapping is accomplished via a memory-based table called the Graphics Address Remapping Table (GART) and used ("walked") by the core logic to perform the re-mapping. In order to avoid compatibility issues and allow future implementation flexibility, this mechanism is specified at a software (API) level. In other words, the actual GART format is not specified; rather a HAL or miniport driver that must be provided with the core logic abstracts it to the API. While this API does not constrain the future partitioning of re-mapping hardware, the re-mapping function will initially be implemented in the chipset or core logic. Note: this re-mapping function should not be confused in any way with the system address translation table mechanism. While some of the concepts are similar, these are completely separate mechanisms, which operate independently, under control of the operating system.

[0063] Understanding the necessity for the Graphics Address Remapping Table ("GART") requires a full understanding of the AGP addressing scheme. Such an understanding can be had from U.S. patent application Ser. No. 08/925,772 referenced above.

[0064] Dual Memory Controllers

[0065] Dual memory controllers process memory requests in parallel, significantly increasing overall memory bandwidth. Other servers using the x86 architecture offer memory bandwidth of either 267 MB/second or 533 MB/second depending on the memory controller chipset used. In contrast, HPSA includes two memory controllers, each with a bandwidth of 533 MB/second. Therefore, total memory bandwidth increases to 1.07 GB/second—two to four times that of other systems.

[0066] Dual memory controllers process memory requests in parallel, significantly increasing overall memory bandwidth. Other servers using the x86 architecture offer memory bandwidth of either 267 MB/second or 533 MB/second depending on the memory controller chipset used. The new servers with HPSA include two memory controllers, each with a bandwidth of 533 MB/second. Therefore, total memory bandwidth increases to 1.07 GB/second; two to four times that of prior art systems.

[0067] Dual peer-PCI buses provide greater I/O bandwidth, the ability to balance system resources and greater system I/O integration and expandability. A single PCI bus provides I/O bandwidth of 133 MB/second, which must be shared by many key peripherals such as the hard drive and NIC. With dual-peer PCI buses, each bus can provide peak bandwidth in parallel with the other controller, allowing an aggregate I/O bandwidth of 267 MB/second. This implementation provides twice the bandwidth of single bus architectures.

[0068] Another benefit of dual-peer PCI buses is the ability to balance system resources across the two buses to gain improved performance. For example, the embedded Wide-Ultra SCSI controller and PCI-based network interface controller can be associated with separate buses. This means that these two highly used peripherals do not have to compete for access to the same PCI bus, which again improves performance.

[0069] Finally, dual-peer PCI buses also allow for greater system I/O integration and expandability by supporting up to 12 PCI devices which is twice the number supported on single bus implementations. Dual peer PCI buses allow the new present invention to deliver more PCI-based I/O expansion slots while also integrating other PCI components, such as the SCSI, on the system board.

[0070] With dual memory controllers and dual peer-PCI buses, the present invention delivers optimized multiprocessing support. Both the Intel Pentium II and Pentium Pro processors provide enhanced multiprocessor support by enabling the multiple processors to share the CPU bus. However, most multiprocessing implementations using the X86 architecture take advantage of this support by simply adding an additional processor to an already existing single processor design. However, HPSA can be used to enhance memory and I/O bandwidth as well. Multiprocessor systems designed without the HPSA will encounter a bottleneck as the multiple processors try to access the other system resources, such as memory and I/O subsystems that have not been enhanced to accommodate the additional data traffic. The HPSA reduces these bottlenecks by enhancing the memory and I/O subsystems with dual memory controllers and dual peer-PCI buses, to accommodate the increased data traffic from multiple CPUs.

[0071] Adding AGP to HPSA

[0072] Adding AGP to a HPSA computer system creates several problems. First, because memory is located on both memory controllers, it is possible that the AGP or the GART physical system memory can be allocated to memory residing on the non-AGP memory controller. While this information can be passed between controllers, such processing

degrades system performance because it consumes both time and valuable host bus bandwidth. It is desirable, therefore, for the operating system to allocate AGP and GART memory to physical system memory that is connected to the same core logic unit as the AGP device in order to reduce the need to contact memory across the host bus. Second, multiple core logic architectures equipped with AGP can quickly lead to computer systems with multiple AGP busses—exacerbating the problem.

[0073] HPSA has many advantages. One of the most important advantages is its ability to utilize all of the system memory **406** (see **FIG. 4**) on all of the various core logic chipsets **404** for any device, dynamically, at runtime. Consequently, memory allocated for a first central processing unit can reside both on the first set of RAM **406***a* as well as on the second set of system RAM **406***b*. While this "pooling" of resource can yield tremendous benefits, it can also increase demands on the host bus **403**, through which all data is transferred between the CPUs **402** and core logic chipsets **404**. This is the problem solved by the present invention. According to the present invention, memory allocated for AGP **410**, for example, is first allocated from system RAM **406***a* thereby avoiding use of the host bus **403** resources and increasing overall system performance.

[0074] To support multiple core logic architectures equipped with AGP, the operating system requires specific information not readily available; such as how many core logic units are present, how much memory resides on each core logic, how is memory interleaved between core logic units, etc. The present invention utilizes one of the advanced configuration power interface (ACPI) control methods that pass along crucial information to the operating system. ACPI is a system regulatory scheme that governs the interfaces between the operating system software, the hardware, and the BIOS software. ACPI also defines the semantics of those interfaces. Intel Corporation, Microsoft Corporation, and Toshiba developed the ACPI specification. The Revision 1.0 ACPI specification was released on Dec. 22, 1996 and is readily available from Microsoft Corporation of Redmond Washington and is incorporated herein by reference for all purposes. Additional information can be found in the "Draft ACPI Driver Interface Design Notes and Reference" which is also available from the Microsoft Corporation of Redmond, Wash. and this document too is incorporated herein by reference for all purposes.

[0075] By employing the ACPI control methodology, a computer system with one or more AGP-equipped core logic chipsets can pass to the operating system (OS) the information required to solve the above-mentioned problems in the prior art and still retain the advantages afforded by HPSA. Specifically, a new ACPI table is developed in the present invention to pass pertinent information to the operating system. The table is called the Memory Bank Address Table (MBAT). Special system BIOS ACPI Source Language (ASL) builds the MBAT dynamically at runtime. Accessed via an ACPI control method, the AGP driver and GART miniport driver use the MBAT information before making calls to the operating system's memory manager. The drivers pass ranges, derived from the MBAT, to the memory manager to, for example, the Windows NT 5.0 operating system routine "MmAllocatePagesForMdl( )." The MmAllocatePagesForMdl( ) routine allows drivers to allocate pages of memory that lie within driver-specified physical address ranges. The memory manager attempts to allocate memory within the specified range. If memory cannot be allocated within the required range, the drivers allocate memory from outside the range as a fallback.

[0076] The Memory Bank Address Table (MBAT) contains all of the information needed by the AGP driver to build a suitable memory descriptor list (MDL) for allocating AGP/GART resources. The MBAT is a predefined structure, format, or record that contains several elements or fields. In the preferred embodiment of the present invention, these elements include a Table Version, the AGP Bus Number, a Valid Entry Bitmap, and a Decode Range array. The Table Version field allows for future changes to the format or the structure of the MBAT. Typically, the Table Version contains an integer value that is incremented by one for each new version. The first Table Version is assigned the number 1.

[0077] In systems that support multiple AGP devices, the Bus Number field provides a means to distinguish between AGP devices if more than one AGP device is present. The value in the Bus Number field is typically the number of the bus behind a PCI-to-PCI bridge device. If the OS is using virtual bus numbers, the AGP driver or GART miniport driver will need to convert the bus number returned by the OS to a virtual equivalent.

[0078] The MBAT has a fixed number of physical address ranges contained in the Decode Range array. The Valid Entry Bitmap is used to indicate the entries that are filled with actual information. Each bit has a one-to-one correspondence with an element in the Decode Range array.

[0079] The physical address ranges are contained within the Decode Range array. Only entries with enabled Valid Entry Bitmap bits contain information. In the preferred embodiment of the present invention, the Decode Range array element has two values, a Begin value and an End value. The Begin value is the starting physical address of an address range. An address can being anywhere in a 64-bit address space as provided by the system architecture. The End value is the physical address that defines an inclusive range from the Begin address, and, therefore defines the physical address range. Obviously, the End address must come after the Begin address in the physical memory region.

[0080] The MBAT is built before the need for memory allocation arises. In the preferred embodiment of the present invention, the MBAT is built dynamically by the system BIOS as needed. Dynamic generation of the MBAT as needed allows for reconfiguration of the computer system without rebooting. This reconfiguration of the computer system while the system is running is often called "hot-swapping" or "hot-plugging." However, in an alternate embodiment where the system configuration never changes, the MBAT may be built once at system startup and simply stored until needed, thereby speeding up somewhat those system memory allocation calls that require the MBAT.

[0081] If the system BIOS does not support the building of the MBAT, the GART miniport driver or the AGP driver can build the MBAT. The structure of the MBAT contains the ranges of memory local to the AGP device depending upon the configuration of the computer system found during startup (POST). The actual form of the MBAT is chipset specific because of the use of certain registers. In any case, by the time that the operating system has booted, the MBAT

will have been created and the operating system will know of the location of the MBAT. The location of the MBAT can be in system memory, in non-volatile RAM (NVRAM), on a hard disk drive, a network drive that is connected to the computer system via a network interface card, or on any volatile or non-volatile memory that can be connected to the computer system which can contain the contents of the MBAT. In the preferred embodiment of the present invention, however, the MBAT will be contained within the system memory.

[0082] The method of building of the MBAT is illustrated in **FIG. 5**. The method is started in step **502**. First, the machine is turned on (or is reset), step **504**. Next, as is typically of most computer systems, the basic input-output system (BIOS) POST is activated. At this point, the BIOS POST program has absolute control over everything in the computer system. Initially, no memory is allocated for any use, except for the first 1 KB that reserved for the interrupt vector table (IVT) and the top of the first megabyte with a ROM containing suitable boot code. During the POST process, some IVT entries are filled in with pointers to interrupt service routines (ISRs) located in the motherboard BIOS. Also, some data is placed in the BIOS area, which is, typically, in the region immediately following the IVT. It is at this point when the MBAT is built, step **508** and placed in NVRAM. When the BIOS POST is completed, it loads the operating system, specifically loading the boot sector program from the boot disk. The boot sector program then loads the actual operating system. Once the operating system is booted, it recognizes the MBAT, step **510**, and the process is completed, step **512**.

[0083] Accessing the MBAT is accomplished through an ACPI control method owned by the AGP device. The AGP driver attempts to invoke the BANK method through the ACPI interface. The value that is returned from the method call is a buffer filled with the MBAT. In the preferred embodiment of the present invention, the size of the buffer is no less than 131 bytes.

[0084] A typical MBAT is shown in Table 1. Each number on the right indicates the byte offset to the table needed by the control method to place data in the appropriate field positions. The ACPI source language (ASL) implementation provided below uses the offset values to create field objects.

TABLE 1

0x01		//	0			
0x01		//	1			
0x05		//	2			
0x0000000000000000	0x000000007FFFFFF	//	3	7	11	15
0x0000000000000000	0x0000000000000000	//	19	23	27	31
0x000000000C000000	0x000000000FFFFFFF	//	35	39	43	47
0x0000000000000000	0x0000000000000000	//	51	55	59	63
0x0000000000000000	0x0000000000000000	//	67	71	75	79
0x0000000000000000	0x0000000000000000	//	83	87	91	95
0x0000000000000000	0x0000000000000000	//	99	103	107	111
0x0000000000000000	0x0000000000000000	//	115	119	123	127

[0085] In the preferred embodiment, the MBAT (as shown in Table 1) contains eleven lines. The first line contains the Table Version field. The second line contains the Bus Number field. The third line contains the Valid Entry Bitmap Field. Finally, the next eight lines contain the Decode Ranges, with the Begin field on the left and the End field adjacent to the right of the Begin field. The various offset numbers are shown on the right of each line, after the "//" characters.

[0086] The ASL builds the MBAT using dynamic information from the memory controller. The memory decode rows are read directly from the memory controller and translated into the MBAT buffer. The AGP parent PCI-to-PCI bridge is queried to get the bus number dynamically. If the OS is using virtual bus numbers, the AGP driver will need to convert the Bus Number to a virtual equivalent.

[0087] The AGP driver and GART miniport driver must invoke the ACPI control method associated with the AGP device to retrieve physical memory information. This is accomplished by issuing, for example, IRP_MJ_PNP and IRP_MN_QUERY_INTERFACE calls to the AGP's physical device object (PDO) to get the ACPI direct call interface. Although the interface is not necessarily needed, a successful query indicates that an ACPI object is associated with the device driver object. Otherwise, there is no reason to attempt to invoke the BANK control method and the AGP driver should resume normal handling for GART allocations. The PDO is an ideal vehicle for this activity because the PDO is constructed for the express purpose of allowing the I/O manager of the operating system and the device drivers to know that status of an I/O device at all times. Device objects make instant status checking possible by keeping information about a device's characteristics and state. There is one device object for each virtual, logical, and physical device on the computer system. Consequently, a PDO for the AGP device will be constructed and available as a matter of course.

[0088] The AGP driver takes the following specific steps to accomplish the task of retrieving physical memory information. First, a reference variable is loaded with the attached device reference (the AGP PDO). Second, an I/O request packet (IRP) is allocated for subsequent use. Third, the query interface (QUERY_INTERFACE method in Windows NT) is used to obtain the address of the direct-call ACPI interfaces. Fourth, the IRP retrieves the ACPI interface. Fifth, the bank control method (e.g., CM_BANK-_METHOD in Windows NT) is invoked to retrieve the MBAT that is needed to build a set of memory descriptor lists (MDL) using the ICTRL_ASYNC_EVAL_METHOD and passing a buffer large enough to hold the MBAT. It is important to note that the memory manager of new operating systems may now support the function MmAllocatePages-ForMdl( ) that was designed for use by AGP drivers to create

MDLs that cover specific bands of physical memory. This support is necessary on systems with multiple memory controllers (such as HPSA). However, the current implementation of MmAllocatePagesForMdl( ) assumes that all of the memory bands are of the same width. The equal-width memory band scenario exists only if the memory banks are populated with identically sized dual inline memory modules (DIMMs). Flexible memory architectures allow variable sized DIMMs and, therefore, renders the banding technique supported by MmAllocatePagesForMdl( ) useless. In that situation, multiple calls must be made to create MDLs that describe the memory bands and those (multiple) MDLs must be combined to form a single MDL.

[0089] The actual memory allocation scheme depends upon how the driver intends to use the memory so allocated. In this instance, the driver need only perform the following steps for the GART memory. It will be understood by those skilled in the art that the exact procedural steps will depend upon the operating system involved and the procedures and methods available in that operating system. In the preferred embodiment as described herein, the Microsoft Windows NT 5.0 operating system is used and the following procedure depends upon that operating system. Once the driver obtains the MBAT, the driver examines the ValidEntryBitmap field to determine which entries in the DecodeRange[] array is valid. Next, for each valid entry, a call is made to MmAllocatePagesForMdl( ) passing a skip value of zero. A combined MDL must be created before attempting to lock the pages. The pages described by the MDL are then locked into memory using MmProbeAndLockPages( ) method. Finally, the pages are mapped into the virtual address space using MmMapLockedPagesSpecifyCache( ) method. The driver must manage multiple MDLs, tiling the mapped ranges to create the contiguous virtual address space needed by the AGP device.

[0090] The general method of memory allocation operation is illustrated in **FIG. 6**. The process is started in step **602**. First, in step **604**, a check is made to determine if a copy of the MBAT is already available to the AGP driver (or GART miniport driver) locally. If so, execution skips to step **612**. Otherwise, execution continues to step **606** where the driver calls the ACPI bank method of the memory manager of the operating system. Next, in step **608**, the memory manager returns a copy of the MBAT to the requesting driver. Optionally, the copy of the MBAT can be stored locally to the requesting driver for quick reference in subsequent memory allocations procedures, step **610**. In step **612**, the AGP driver (or GART miniport driver) references the MBAT to determine if any ranges of local memory (i.e., memory connected to the same core logic chipset as the AGP device) is available. The local memory so identified, as much as needed, and the ranges of those memory locations are used to generate parameters for the call to the memory manager for the allocation of new memory (for example, the MmAllocatePagesForMdl( ) in Windows NT). If the call is successful, step **616**, then the AGP device can use the newly allocated local memory in the normal fashion. If the call was not successful, the AGP driver (or GART miniport driver) can make the same call to the memory manager without requesting specific ranges (i.e., allocate memory from any available resources within the computer system). Finally, the process ends in step **618**.

[0091] A special situation arises for those operating systems that do not accommodate specific ranges in the memory allocation request. In those cases, the method described above can be modified to suit the situation. Such a method is illustrated in **FIG. 7**. First, the method is started, step **702**. Next, in step **704**, the AGP driver or GART miniport driver checks to see if it has a copy of the MBAT. If so, execution skips to step **716**. Otherwise, execution continues to step **706**, where the AGP driver (or GART miniport driver) invokes the ACPI bank method call to the operating system. The operating system returns the MBAT, step **708**, and the requesting driver stores the MBAT locally, step **710**. In step **716**, the memory manager of the operating system is called, requesting that new memory be allocated. It is at step **718** where this alternate embodiment differs from the preferred embodiment. In step **718**, the newly allocated memory range that was returned by the memory manager of the OS is compared to the MBAT. The comparison of step **718** is used to determine if the newly allocated memory is physically located within the same core logic chipset as the AGP device. In contrast, in the preferred embodiment, if the pages cannot be allocated within the specified range (i.e., local memory), then the method call returns an error and the preferred method then allocates memory anywhere, without restriction, as a fall back (see step **616** of **FIG. 6**). In this alternate embodiment, if the newly allocated memory is local (i.e., step **718** is positive), then step **720** is executed. In step **720**, a check is made to determine if sufficient memory has been allocated. If not, execution returns back to step **716** so that more memory can be allocated. Otherwise, the non-local memory is released (while the local memory is retained) in step **722** and execution ends in step **722**. Once the memory is allocated, the AGP driver or GART miniport driver utilizes the memory in the normal manner.

[0092] The present invention, therefore, is well adapted to carry out the objects and attain the ends and advantages mentioned, as well as others inherent therein. While the present invention has been depicted, described, and is defined by reference to particular preferred embodiments of the invention, such references do not imply a limitation on the invention, and no such limitation is to be inferred. The invention is capable of considerable modification, alternation, and equivalents in form and function, as will occur to those of ordinary skill in the pertinent arts. The depicted and described preferred embodiments of the invention are exemplary only, and are not exhaustive of the scope of the invention. Consequently, the invention is intended to be limited only by the spirit and scope of the appended claims, giving full cognizance to equivalents in all respects.

What is claimed is:

1. A computer system, said computer system comprising:

a central processing unit connected to a host bus;

a system memory having an addressable memory space comprising a plurality of bytes of storage, wherein each of said plurality of bytes of storage has a unique address;

a first memory controller connected to said host bus, said first memory controller connected to a first portion of said system memory;

a second memory controller connected to said host bus, said second memory controller connected to a second portion of said system memory;

an input-output bus connecting an input-output device to said first memory controller and to said host bus, said input-output device being controlled by a driver; and

a memory bank allocation table, said memory bank allocation table capable of containing said unique addresses of said first portion of said system memory;

wherein said driver references said memory bank allocation table in order to determine a range of said system memory connected to said first memory controller so that said driver may allocate some of said first portion of said system memory.

2. A computer system as in claim 1, wherein said memory bank address table contains a version number.

3. A computer system as in claim 1, wherein said memory bank address table contains a bus number.

4. A computer system as in claim 1, wherein said memory bank address table contains a valid entry bitmap.

5. A computer system as in claim 1, wherein said memory bank address table contains an array of decode ranges.

6. A computer system as in claim 5, wherein said decode ranges of said memory bank address table each contain a begin field.

7. A computer system as in claim 5, wherein said decode ranges of said memory bank address table each contain an end field.

8. A computer system as in claim 1, wherein said memory bank address table is created by BIOS.

9. A computer system as in claim 1, wherein said memory bank address table is created by a GART miniport driver.

10. A computer system as in claim 1, wherein said memory bank address table is created by an AGP driver.

11. A computer system as in claim 1, wherein said driver is a GART miniport driver.

12. A computer system as in claim 1, wherein said driver is an AGP driver.

13. A method of building a memory bank address table, said method comprising the steps of:

activating a BIOS, said BIOS capable of building said memory bank address table; and

building said memory bank address table in a memory location.

14. A method as in claim 13 wherein said memory location is system memory.

15. A method as in claim 13 wherein said memory location is a non-volatile random access memory.

16. A method of building a memory bank address table, said method comprising the steps of:

activating a driver, said driver capable of building said memory bank address table; and

building said memory bank address table in a memory location.

17. A method as in claim 16, wherein said driver is a GART miniport driver.

18. A method as in claim 16, wherein said driver is an AGP driver.

19. A method of allocating local memory for an AGP device, said method comprising the steps of:

(a) providing an operating system;

(b) providing a driver;

(c) calling an ACPI method of said operating system with said driver;

(d) retrieving a memory bank address table from said operating system;

(e) reading said memory bank address table with said driver in order to determine ranges of system memory local to said AGP device; and

(f) allocating system memory local to said AGP device based upon said ranges determined in said step (e).

20. A method as in claim 19 wherein said driver is an AGP driver.

21. A method as in claim 19 wherein said driver is a GART miniport driver.

22. A method of allocating local memory for an AGP device, said method comprising the steps of:

(a) providing an operating system having a memory manager;

(b) providing a driver;

(c) calling an ACPI method of said operating system with said driver;

(d) retrieving a memory bank address table from said operating system;

(e) calling said memory manager to allocate a range of system memory for said AGP device;

(f) comparing said range of system memory allocated in said step (e) with said memory bank address table to determine if said range of system memory is physically located on a core logic chipset connected to said AGP device; and

(g) repeating said steps (e) and (f) until sufficient system memory has been allocated.

23. A method as in claim 22, said method further comprising the step of:

(h) releasing system memory determined not to be local to said AGP device in said step (f).A method as in claim 22 wherein said driver is an AGP driver.

24. A method as in claim 22 wherein said driver is a GART miniport driver.

26. A computer system having at least two memory controllers which connects a computer processor and memory to an input-output bus, said input-output bus connected to at least one input-output device, said system comprising:

a system processor executing software instructions and generating data;

a system memory having an addressable memory space comprising a plurality of bytes of storage, wherein each of said plurality of bytes of storage has a unique address;

said software instructions and said data being stored in some of said plurality of bytes of storage of said system memory, wherein said data are stored in a plurality of pages of data, each of said plurality of pages of data comprise a number of said plurality of bytes of storage;

a core logic chipset having an accelerated graphics port (AGP) adapted for an AGP processor, wherein said AGP processor generates video display data from said graphics data for display on a video display;

said core logic chipset having a first interface logic for connecting said system processor to said system memory;

said core logic chipset having a second interface logic for connecting said system processor and said system memory to said AGP; and

a graphics address re-mapping table (GART) having a plurality of entries, each of said plurality of GART entries comprising an address pointer to a corresponding one of said plurality of pages of graphics data and feature flags for customizing said corresponding one of said plurality of pages of graphics data, wherein said core logic chipset uses said plurality of GART entries for re-mapping said plurality of pages of graphics data into an AGP device address space for use by said AGP processor in generating said video display data, and said feature flags for customizing the operation thereof.

* * * * *